Optimizing TEE Protection by Automatically Augmenting Requirements Specifications

Siddharth Dhar

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science

Eli Tilevich, Chair
Osman Balci
Na Meng

May 13, 2020
Blacksburg, Virginia

Keywords: Trusted Execution Environment, Natural Language Processing, Recommendation System, Critical Program Information, Requirements Specification

Optimizing TEE Protection by Automatically Augmenting Requirements Specifications

Siddharth Dhar

(ABSTRACT)

An increasing number of software systems must safeguard their confidential data and code, referred to as critical program information (CPI). Such safeguarding is commonly accomplished by isolating CPI in a trusted execution environment (TEE), with the isolated CPI becoming a trusted computing base (TCB). TEE protection incurs heavy performance costs, as TEE-based functionality is expensive to both invoke and execute. Despite these costs, projects that use TEEs tend to have unnecessarily large TCBs. As based on our analysis, developers often put code and data into TEE for convenience rather than protection reasons, thus not only compromising performance but also reducing the effectiveness of TEE protection. In order for TEEs to provide maximum benefits for protecting CPI, their usage must be systematically incorporated into the entire software engineering process, starting from Requirements Engineering. To address this problem, we present a novel approach that incorporates TEEs in the Requirement Engineering phase by using natural language processing (NLP) to classify those functional requirements that are security critical and should be isolated in TEE. Our approach takes as input a requirements specification and outputs a list of annotated functional requirements. The annotations recommend to the developer which corresponding features comprise CPI that should be protected in a TEE. Our evaluation results indicate that our approach identifies CPI with a high recall and passable precision to incorporate safeguarding CPI into Requirements Engineering.

Optimizing TEE Protection by Automatically Augmenting Requirements Specifications

Siddharth Dhar

(GENERAL AUDIENCE ABSTRACT)

An increasing number of software systems must safeguard their confidential data like passwords, payment information, personal details, etc. This confidential information is commonly protected using a Trusted Execution Environment (TEE), an isolated environment provided by either the existing processor or separate hardware that interacts with the operating system to secure sensitive data and code. Unfortunately, TEE protection incurs heavy performance costs, with TEEs being slower than modern processors and frequent communication between the system and the TEE incurring heavy performance overhead. We discovered that developers often put code and data into TEE for convenience rather than protection purposes, thus not only hurting performance but also reducing the effectiveness of TEE protection. By thoroughly examining a project's features in the Requirements Engineering phase, which defines the project's functionalities, developers would be able to understand which features handle confidential data. To that end, we present a novel approach that incorporates TEEs in the Requirements Engineering phase by means of Natural Language Processing (NLP) tools to categorize the project requirements that may warrant TEE protection. Our approach takes as input a project's requirements and outputs a list of categorized requirements defining which requirements are likely to make use of confidential information. Our evaluation results indicate that our approach performs this categorization with a high degree of correctness to incorporate safeguarding the confidentiality related features in the Requirements Engineering phase.

Dedication

To the healthcare and essential workers helping to contain this pandemic.

May the force be with you.

Acknowledgments

I would like to thank my advisor Dr. Tilevich for his guidance, support, and efforts, without which this project would not have been possible. Your jovial nature was especially helpful during times when I was stressing out about my graduation. I would also like to thank Dr. Balci for his inputs and helping us by providing the required data for testing the study. I want to thank Dr. Meng for her efforts and suggestions which were invaluable for the completion of this study. I am indebted to the Computer Science Department for the opportunity to pursue and fund my Master's degree at Virginia Tech, and Sharon for helping out with all administrative issues and queries.

I would like to express my gratitude to Yin Liu for his ideas and feedback that were extremely important in helping me complete the project and the associated research paper. I am grateful to Palakh and Nicole for their help in completing the project and this thesis. Lastly, a big thank you to my friends Judee, Sose, Maanav, and Deeksha for their belief and support over the last couple of years.

Contents

List of Figures

List of Tables

List of Abbreviations

CBOW Continuous Bag Of Words

CPI Critical Program Information

EPC Enclave Page Cache

KMP Knuth Morris Pratt

ML Machine Learning

NLP Natural Language Processing

PII Personally Identifiable Information

SGX Software Guard Extensions

TCB Trusted Computing Base

TEE Trusted Execution Environment

Chapter 1

Introduction

Modern information systems, in domains ranging from social media to electronic commerce, store and manipulate private and confidential data, including personally identifiable information (PII). These services become the prime target of security attacks that attempt to gain unauthorized access to sensitive data. For example, Facebook has been one of the biggest targets for hackers trying to obtain access to user accounts [30], with widespread mediate coverage about a recent successful data breach [10]. In addition, as technology advances, an increasing number of systems integrate proprietary algorithms and other intellectual property, sensitive information that must be protected. It is imperative that confidential information be stored securely, so it would not be leaked to the adversary. These realities call for additional measures that can ensure data safety as well as prevent leaks and intellectual property theft [18, 19, 23].

The aforementioned examples represent a small subset of software systems, for which mandating critical program information (CPI) is paramount. Developers strive to provide at least a minimal level of security for the confidential information by employing encryption or hashing mechanisms. Despite these measure, most systems are prone to attacks and have some degree of vulnerability[27].

In the bid to safeguard CPI, one promising technology is trusted execution environments (TEEs). A TEE provides a protected processing environment with its own memory and storage, running in a separate kernel to ensure isolation. Popular TEE hardware, such

as Intel SGX[12] and OP-TEE[3], provide the capability to place code and data in so-called 'secure world' [1] for secure processing. Apple uses another implementation of the TEE specification called *Secure Enclave* [4] to implement and store the data for its Touch ID and Face ID features. In this work, we focus our investigation on the Intel SGX platform due to its wide industry adoption.

TEEs have limited processing capacities and high execution costs. Hence, it would be infeasible to place all code in the secure enclave, as multiple function calls from the TEE and frequent communication between the TEE and the outside world would destroy the performance[24][45]. To safeguard CPI without crippling system performance, developers must place in TEE only those parts of the system's code and data that really need to be protected. In other words, ensuring that the Trusted Computing Base (TCB) is exactly of the required size is key to achieving the best possible tradeoff between security and performance.

Nevertheless, our analysis of a representative sample of open-source projects that use TEEs has revealed a counter-intuitive trend: developers tend to overuse TEEs by isolating non-CPI. As we discuss later in the manuscript, there is a mismatch between the optimal usage of TEEs and how they are currently utilized. One reason for this mismatch is a lack of standards pertaining to using TEEs to safeguard CPI. Another reason is that the issue of taking advantage of TEEs is postponed until the implementation or the maintenance phase in the software development life cycle. This postponement can cause excessive redesign and refactoring as the only avenues for ensuring effective TEE protection.

These realities identify a need to enforce TEE protection standards in the initial phases of the software development life cycle, starting from Requirements Engineering. Subsequent to the requirements gathering phase, these standards can then help the developer determine the

[1] The *outside* and *secure* worlds are standard TEE terms. Code and data isolated in the secure world are protected, and unprotected and compromisable otherwise.

program features that should be protected in the TEEs. To that end, this paper contributes a novel approach that accepts an existing set of software requirements and modifies them with an additional parameter that assists the developer in identifying if a program feature should be isolated in the TEE. We leverage advanced NLP techniques to identify relevant words from a given software requirement by comparing them with a previously identified list of keywords. The relevant words are then applied to determine which features should be protected in the TEE. Specifically, the decision making process is guided by similarity scores. Given a list of classified software requirements, developers can then proceed to the subsequent software development phases while being cognizant of the features identified as needing TEE protection. This systematic approach can reduce the need to refactor existing TEE-dependent code in order to change what comprises CPI that must be isolated in TEE.

The main contributions of this study are:

- We build a set of features that make use of TEE protection based on our analysis of the source code isolated within the TEE in Intel SGX based projects on GitHub.
- A novel approach that makes use of these identified features along with advanced NLP techniques to help classify software requirements. We use fastText word embeddings trained on 4972 research papers from the security and privacy domain.

Next, we discuss the required background information (Section 2.1), then, we define our methodology for the initial study of existing projects on GitHub and the recommendation tool we developed to identify features to be placed in the TEE (Section 4), and present our results (Section 5) followed by a discussion of the limitations of this paper and future work (Section 7).

Chapter 2

Background

In this chapter, we introduce the technical background required to understand the research contributions of this thesis.

2.1 Trusted Execution Environments

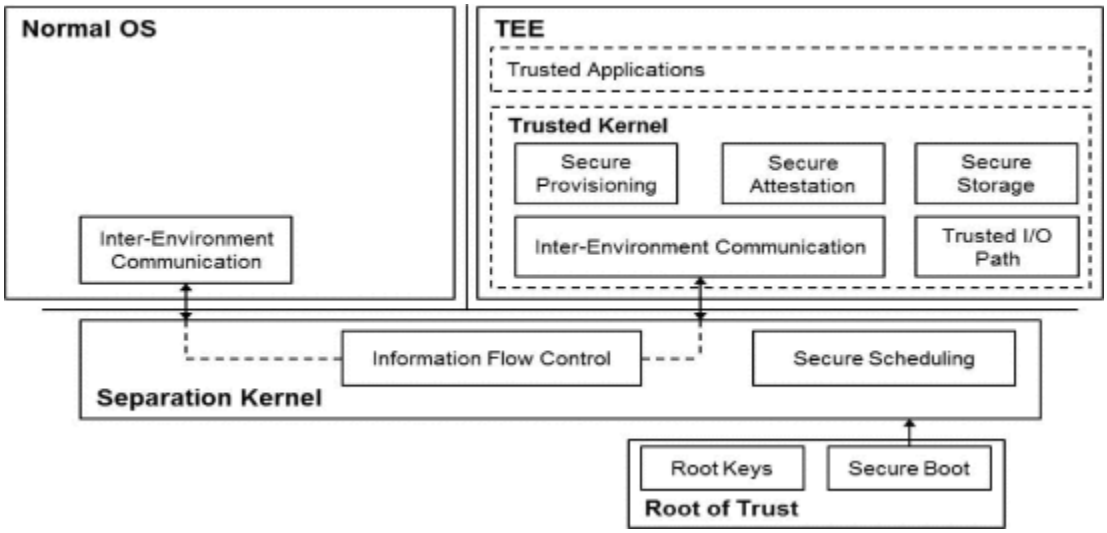

Figure 2.1: TEE building blocks [39]

A TEE is an isolated execution environment that provides secure code execution and data storage capabilities that runs in parallel to the operating system. Since TEEs have their own memory and storage, they are capable of functioning independent of the rest of the CPU. The trusted applications are protected from the other applications on the operating system

be means of hardware isolation, while having the potential to utilize the system processor to its full capability. The applications running inside a TEE are isolated from one another by means of software and cryptography. The authors of [39] describe a TEE as "secure, integrity-protected processing environment, consisting of memory and storage capabilities".

Fig 2.1 shows the different blocks associated with a TEE environment. The separation kernel is essentially a combination of hardware, firmware and software based security kernel that helps simulate a distributed environment thus ensuring isolation for the TEE. It partitions the processing environment and acts as an interface between these partitions facilitating communication between the *secure world* and the *outside world* by means of inter-environment communication modules. The secure scheduling module maintains coordination between the operating system and the TEE ensuring that trusted applications do not significantly lower the overall system performance. Within the TEE, the trusted kernel provides additional modules such as secure storage and trusted I/O.

2.1.1 Intel Software Guard Extensions

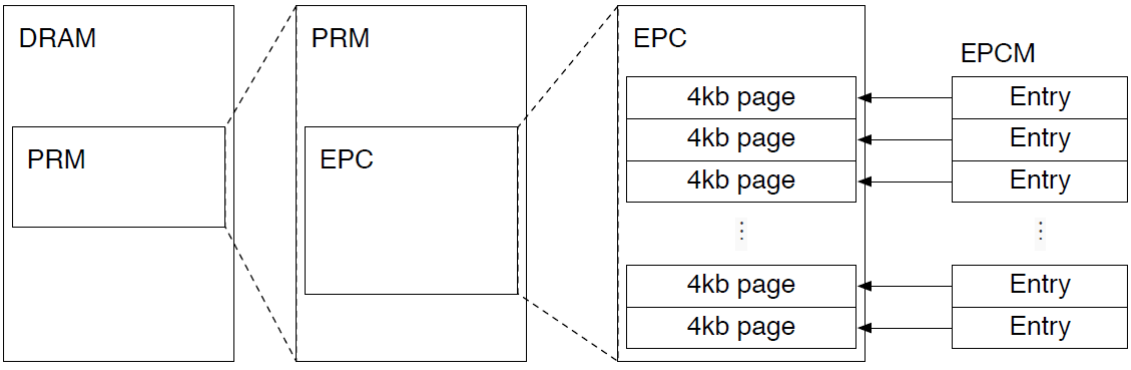

Figure 2.2: Storage of enclave code in memory [12]

Intel's implementation of TEE relies on security related instruction codes that allow creation

of secure 'enclaves' within modern Intel CPUs, for processing trusted applications. It employs a threat model in which all outside processes, even the operating system is not trusted. To secure the process, it encrypts a part of the memory which is then decrypted during runtime within the enclave and only for code and data executing within the enclave needed at that point. As shown in fig 2.2, the code and data associated with the enclave are stored in the Enclave Page Cache (EPC). The EPC itself is contained within the Processor Reserved Memory (PRM) which further is a subset of the DRAM. The PRM cannot be accessed by software and the operating system, thus ensuring isolation for the enclave.

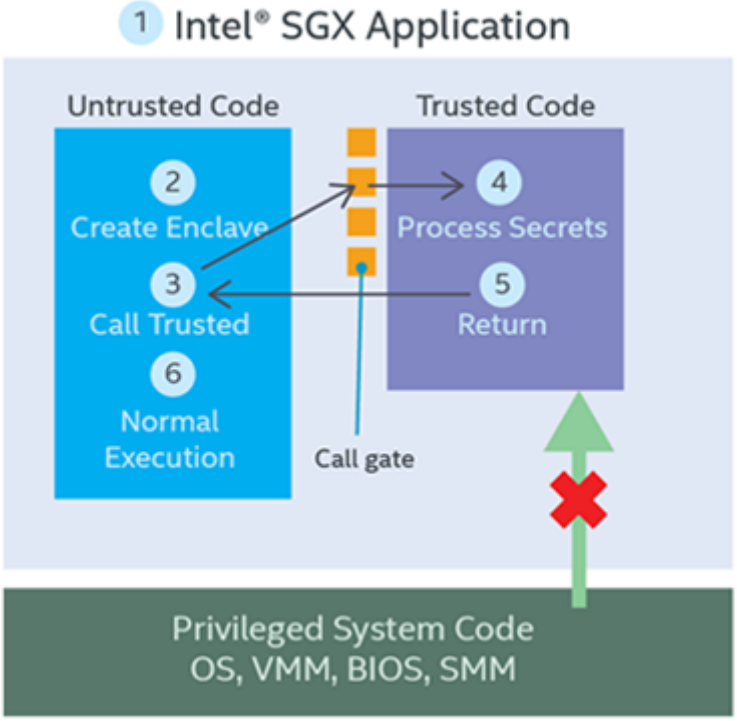

Figure 2.3: SGX application execution flow [20]

As seen in fig 2.3, an SGX application is divided into trusted and untrusted code. The trusted code, that is placed inside the enclave is isolated from the rest of the application and even the operating system. The remaining application constitutes the untrusted code is stored and executed in the *outside world*. Whenever the application needs to make use of

it's secure features, it interacts with the enclave using secure method calls and the enclave returns the results after securely processing the instructions, thus isolating the piece of code that needs to be kept secure.

```
01 enclave {
02
03     trusted {
04         public void store_secret([in, string] char *msg);
05         public int print_hash([out] sgx_status_t *error);
06     };
07     untrusted {
08         void o_print_hash([in] unsigned char hash[32]);
09     };
10
11 };
```

Figure 2.4: Example edl code [7]

An *Enclave Definition Language (edl)* file acts as the interface between the outside world and the enclave. To access code stored in the enclave from the outside world, the untrusted code makes use of *enclave calls (ecalls)* and the enclave then uses *outside calls (ocalls)* to return back to the CPU from the enclave. The edl file contains the definitions to the *ecalls* and *ocalls* as described in an example in Fig 2.4.

2.1.2 Op–TEE

The Op–TEE is another implementation of TEE designed for unsecure Linux kernels that run on Arm architecture[26]. Though initially designed to be compatible with only ARM TrustZone isolation mechanism, it is now capable of running alongside any isolation technology.

Like SGX, Op–TEE too allows for code to be split into secure and unsecure zones. The unsecure code is executed in the operating system kernel termed as *Rich Execution Environment*,

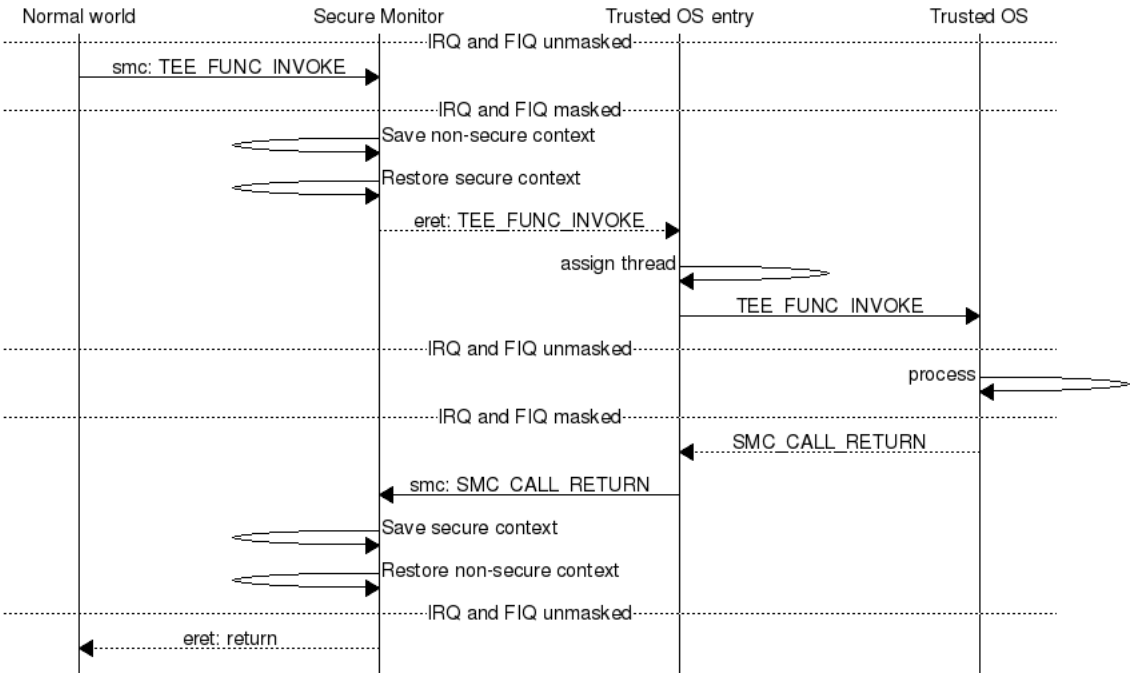

Figure 2.5: Op–TEE entry into secure world [2]

while the secure code runs in the *secure world*.

It makes use of SMC exceptions and interrupt notifications to switch context between secure and normal worlds as depicted in Fig 2.5 and Fig 2.6.

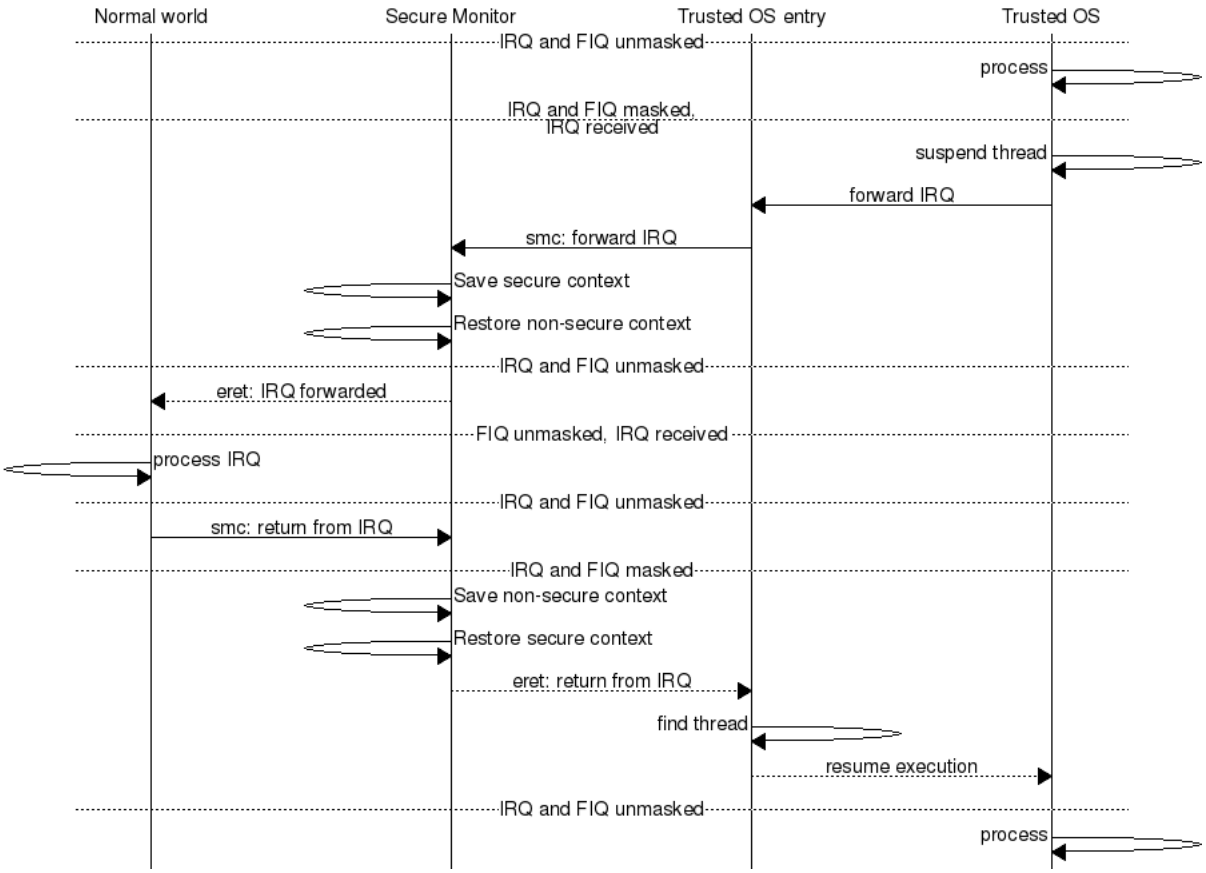

Figure 2.6: Op–TEE leaving secure world to enter normal world [2]

2.2 Word Embeddings

Word Embeddings are a feature representation technique used in NLP to help machines understand words. These embeddings are developed by considering the context of a word with respect to its usage in a given corpus. Considering that these representations are typically developed for a particular domain and are capable of representing the features of each word in the form of vector representations, they have proven to be extremely effective in NLP. Here we describe a few of the most commonly used word embeddings.

2.2.1 Word2Vec

The authors of [29] proposed the first ever word embeddings in the form of *Word2Vec* in 2013 using shallow neural networks. It is capable of representing the linguistic context of words in the training vocabulary in the form of high dimensional word vectors. Each unique word in the corpus is placed based on it's context, leading to similar words being placed close to each other in the vector space.

Word2Vec can compute these word vectors using either the continuous bag of words (CBOW) [22] or the skip–gram [13] model. CBOW relies on the given context to predict the most likely word, whereas the skip–gram model predicts the context based on the given word. However, it does not take into consideration the information available by statistical analysis of the given training data, e.g. frequency of words that occur together and also has low performance in cases when words that are not present in the training vocabulary are used.

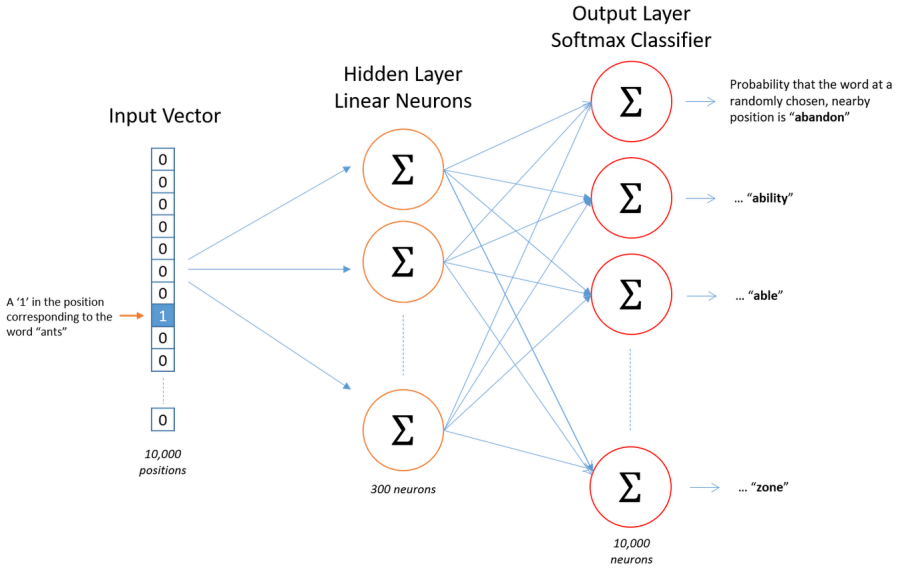

Figure 2.7: Sample 2-layer Word2Vec architecture [5]

2.2.2 GloVe

In a bid to improve upon the existing Word2Vec model, the authors of [33] proposed GloVe embeddings.

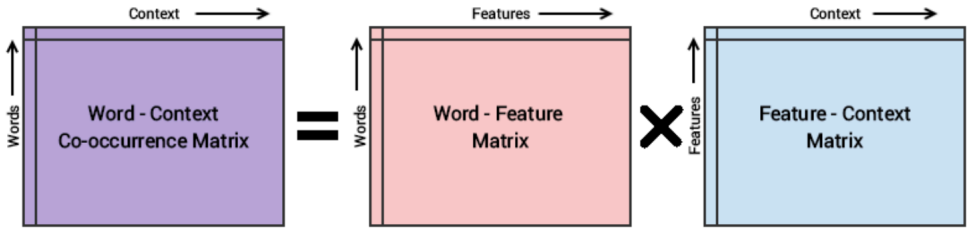

Figure 2.8: Sample GloVe architecture [40]

It can build word vectors using either the local context window or the global matrix factorization method. The local context window uses CBOW and skip–gram the same way as Word2Vec. However, using the global matrix factorization technique, it can take into account the co-occurence frequency if the words. It then reduces term frequency matrices that contain data relating to the presence or absence of words, using linear algebra.

However, these GloVe embeddings are also not capable of performing well when using words not included in the training vocabulary.

2.2.3 fastText

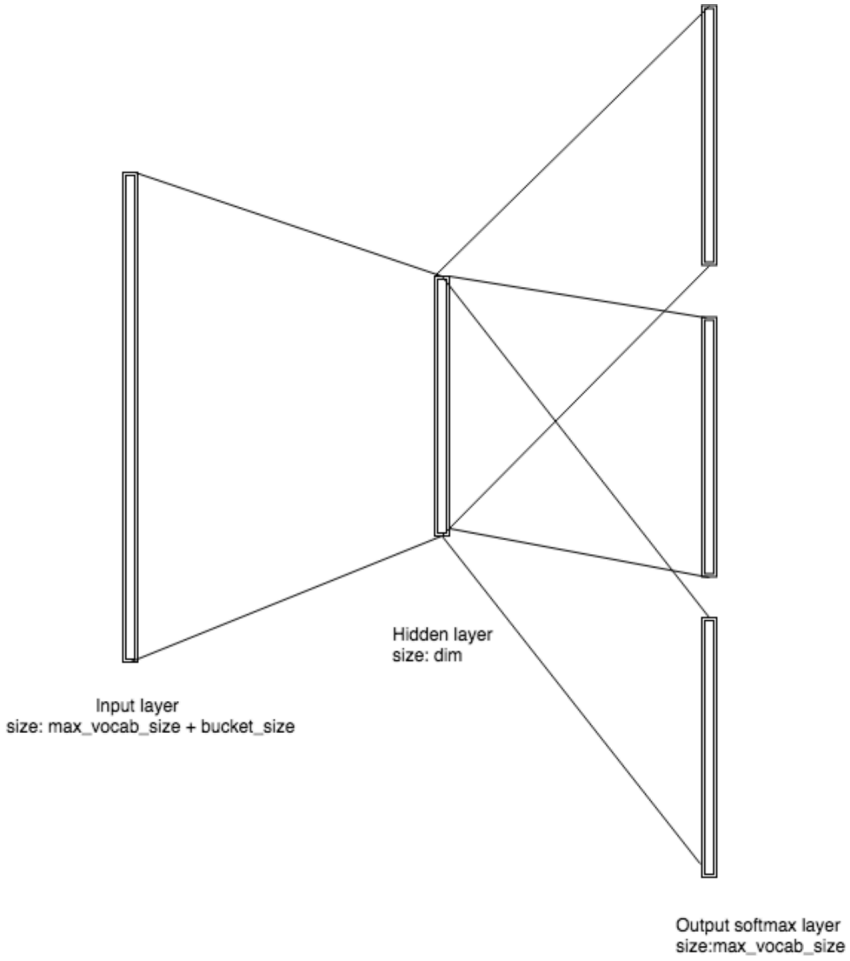

Figure 2.9: Sample fastText architecture [43]

In order to overcome the limitation of out of vocabulary words in the above models, Facebook's AI Reasearch came up with fastText [15] word embeddings. Instead of just considering words during training, it also takes into account the characters, based on a pre-set window size.

Even though the resultant embeddings resemble those given by Word2Vec, the fastText embeddings comprise of a combination of several lower-level embeddings. This allows the fastText models to be trained on significantly lesser amount of data that Word2Vec.

Chapter 3

Literature Review

In this section we discuss the prior works related to our study and tools that we used.

3.1 KMP Algorithm

Knuth, Morris and Pratt propose a string searching algorithm [21] capable of searching for the occurrences of a test sub-string within a given string. They make use of a pre-computed partial match table to improve upon the efficiency of the brute force search, by using data from the table to compute the next point of comparison whenever a mismatch occurs. Our approach also relies on matching keywords in the test software requirements, but in contrast to the KMP algorithm, our tool is able to incorporate synonyms and words similar to a keyword in the search, rather than looking for exact matches. This ensures that even though some requirements may use different vocabulary, the tool is able to extract its correct usage of CPI.

3.2 Classifying Software Requirements

Sharma proposes a classification technique in [41], this uses organizational semiotics [16] to first identify software requirements from existing documentation. The set of class labels are identified using grounded theory [42]. Next, they manually annotate a large corpus of

software requirements from information system based projects with these class labels. Then they train a multi-label machine learning classification model with these labelled software requirements. Extending their study to a different set of classification labels would require the need to re-label the software requirements used for training. Considering the high costs associated with manually labelling datasets, we employed unsupervised NLP techniques to demonstrate the effectiveness of classifying software requirements to help identify features that may benefit from TEE protection. While we propose a classification model as a future enhancement for our research, we could use the proposed tool as an alternative to manual labelling the dataset. Thus, we did not further explore the machine learning classification approach due to lack of available labelled software requirements to train an effective classification model.

3.3 RT-Trust

RT-Trust[24] is an automated program analysis and refactoring framework for existing software systems to isolate their CPI in a TEE, while ensuring that the system continues to adhere to its real-time execution constraints. Its programming model entails developers annotating the functions that are to be isolated in a TEE. A custom LLVM pass then compiles the code and transfers the annotated functions to execute in a TEE. Although this approach alleviates the development burden, its effectiveness with respect to safeguarding CPI depends entirely on the developer's discretion with respect to which features are to be protected. Whether a function comprises CPI and should be isolated in TEE is hard to determine precisely, particularly if the maintenance programmer is not the same who wrote the original code.

3.4 Recommended Usage of TEE

Intel makes a few recommendations for using SGX effectively by presenting representative use cases [28], with another set of recommendations for using trusted execution in different use cases presented in [39]. These recommendations primarily focus on applying SGX to protect sensitive data and attestations. They also describe how TEEs can be applied to secure communication.

3.5 Tika-Python

To build the recommendation tool, we needed to train custom word embeddings specific to the security and privacy domain. As input to train these embeddings, we gathered a large set of research papers in PDF format from *arxiv.org*. We used Tika-Python, the Python library for Apache Tika REST services for this. Apache Tika is a Java based content analysis toolkit which is capable of extracting text and metadata from PDF files. It is also capable of extracting text from images using Tesseract OCR tool.

3.6 t–SNE

t–Distributed Stochastic Neighbor Embedding (t-SNE) [25] is an approach that allows the reduction of high dimensional data to a lesser number of dimensions. Using this approach, it is possible to visualize high dimensional data in 2 or 3 dimensional plots. Scikit–learn's t–SNE implementation [32] calculates joint probabilities using similarities between data points and then tries to minimize the difference between the high dimensional and low dimensional data. To speed up the computations and lower the noise in the input data, it utilizes

traditional dimensionality reduction methods before computing the joint probabilities.

3.7 Evaluation Metrics

To evaluate the results obtained by classifying the software requirements in the test dataset, we used some of the standard techniques for evaluating classification performance: accuracy, precision, recall, and F-1 score. Among these metrics, precision, recall, and F-1 score are often used for skewed datasets.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{3.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.3}$$

$$F - 1\ Score = \frac{2 * (Precision * Recall)}{Precision + Recall} \tag{3.4}$$

Chapter 4

Analysis and Application of TEE Usage

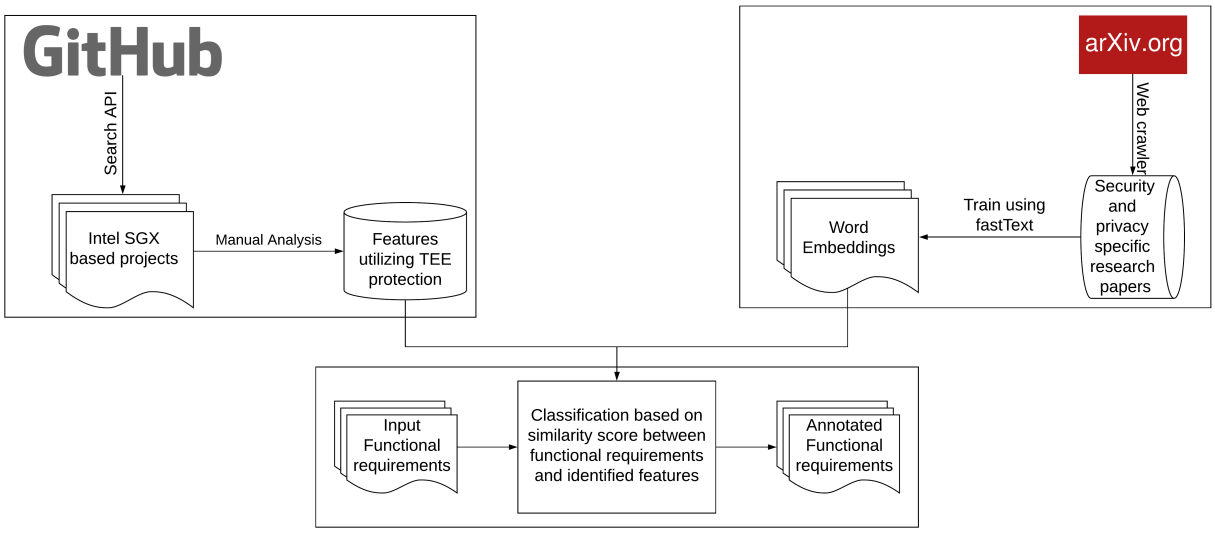

Figure 4.1: High-Level overview of project methodology

Fig 4.1 describes the high level overview of the study. First, we analyzed a large set of GitHub repositories to identify how TEEs are used in open-source software project. Then, based on our findings, we created a recommendation tool to help identify those software requirements that can benefit from TEE protection. We describe these two parts in turn next.

4.1 Analysing TEE Usage in GitHub Repositories

To obtain a realistic snapshot of the common usage of TEE (we focus on Intel SGX) in open-source software, we used GitHub as one of the most popular and largest repositories.

4.1.1 Gathering Repositories

To retrieve those repositories whose source code integrates Intel SGX, we used GitHub's search API[6]. Specifically, we searched for the repositories in which any source file includes the *sgx_urts.h* header. Before running any secure code, a SGX-based project must first create an enclave by invoking the *create_enclave()* function, whose header is in *sgx_urts.h*. To automate the process, we created a Python script that automatically fetches the repositories that match the aforementioned search criteria as shown:

```
from github import Github
g = Github("#GitHub token")
query = f'"{"sgx_urts.h"}" in:file extension:cpp'
result = g.search_code(query, order='desc')
```

The script first creates a list of URLs of the matched GitHub repositories. We collected a total of 429 repositories. A detailed description of this step appears in Section 5.

4.1.2 Analyzing Open-Source Projects

As recommended by Intel[28] and other guidelines [1, 39], we categorized the projects to identify if they use TEEs optimally. The parameters used in this identification process are based on the set of features that were put inside the enclaves and the projects' TCB ratio.

We analyzed each repository to identify the features that incorporate SGX enclaves. The analysis places its subjects into one of the following categories:

1. *Not Relevant*–a repository contains no security or privacy related features in the enclaves, suggesting that these features can be moved out.

2. *Needs Refactoring*–a repository contains an enclave whose content contains code that could have been placed in the outside world.

3. *Relevant*–a repository's enclaves contains the code of only security or privacy related features.

4. *N/A*–a repository does not use TEEs, but is a wrapper for a TEE or a library that makes use of TEEs, or a benchmark tool. However, the project itself is not a TEE based project.

In addition to the aforementioned categorization, we also extracted and persisted the features placed in the enclaves, so they can be analyzed to determine the most relevant features that use TEE protection. We used these features to identify a set keywords that we apply in the second phase of our study as a mechanism for identifying those software requirements that contain security or privacy related features.

4.1.3 Calculating the Ratio of TCB

In addition to accurately identifying features that should be protected in the TEE, it is also important to ensure that the ratio of the resulting trusted computing base (TCB) to the complete project is as small as possible. As TEE hardware is known to be slower than the main CPUs, large TCBs can incur a high performance overhead. Besides, invoking

each TEE-based functionality is expensive, due to the significant communication overhead between the CPU and the TEE.

To calculate the TCB ratio for the analyzed projects, we used the Python library *pygount*[37]. This library enables us to get the total number of lines of code in the projects. Then, we found all the Enclave Definition Language (edl) files which have a *.edl* extension in the project and stored the directory in which the *edl* file is placed. These files are the interface to the enclave, and all enclave code is most likely placed in the same directory as the *edl* file. Once we had the directories of the *edl* files, we again used *pygount* to count the lines of code within these directories to get the lines of code in the enclave as shown below:

```
def countLines(path) :
    c1 = 0
    paths = os.walk(path)
    for x in paths:
        for files in x[2]:
            try:
                analysis = pygount.source_analysis(x[0]+'/'+files, 'test')
            except:
                with open("errors.txt", "a") as errorFile:
                    errorFile.write(x[0]+'/'+files+'\n')
            else:
                c1 += analysis.code
    return c1
```

The TCB size was then calculated as

$$TCB\ Ratio = \frac{Enclave\ lines\ of\ code}{Total\ lines\ of\ code} \tag{4.1}$$

4.2 Automated Annotation of Software Requirements

The findings of the 'GitHub Repository Analysis' phase reveal that 61.3% of the GitHub repositories that use Intel SGX, either do not use the enclave for any security or privacy related features, or include code in the enclave that does not cater to any CPI mixed with the features that actually need to be isolated in the enclave. Motivated by these results, we developed a classification approach that takes a project's requirements specification as input and returns an annotated list of software requirements that informs the developer which requirements involve CPI. This classification of software requirements can help incorporate an awareness of TEE in the initial phases of the software development life cycle and also save the additional overhead of refactoring completed implementations. This approach also provides a more comprehensive view of the system as a final product, before the development phase begins, thus enabling the developer to leverage these annotated requirements in order to properly isolate CPI during the development phase.

We used the identified features reported by the 'GitHub Repository Analysis' phase to develop a recommendation system that accepts a project's requirements specification and classifies the software requirements as 'Relevant to TEE' or 'Not Relevant to TEE'. Given a set of requirements for a new project, we used an NLP-based technique with *fastText* word embeddings to compute the similarity scores between the word tokens extracted from the project's requirements and the CPI-specific features of the previously analyzed projects. For this task, we used the *pyfasttext* Python library[44] as shown:

```
from pyfasttext import FastText
model = FastText('model.bin')
```

Then we experimentally determined what should be the optimal similarity score to serve as the threshold for guiding the classification process. Specifically, whenever the similarity score crossed the determined threshold, the corresponding software requirement was classified as specifying a CPI-related feature (i.e., 'Relevant to TEE').

4.2.1 Initial Approach

As discussed in Section 2.2, word embeddings can effectively represent textual data in various NLP applications. So our automated approach takes advantage of word embeddings.

As an initial experiment, we used the pre-trained *fastText* word embeddings[17] for the English language as trained on the English Wikipedia corpus. The pre-trained model consists of 300 dimensional representation of words using a continuous bag of words (CBOW) model and character n-grams of length 5. We used this pre-trained model to get word vectors for the identified keywords and the word tokens from the software requirements.

We then performed multiple experiments with these word embeddings. We varied the threshold value between 0.65 - 0.8 in steps of 0.05 and achieved the best results with the threshold of 0.75. However, the obtained results proved less than satisfactory. To improve them, we trained our own word embedding models[8] on a corpus specific to the security and privacy domain, so as to be able to work with a more context-relevant vocabulary. We then performed similar experiments with the new word embeddings and compared the results with those of the pre-trained model. Section 5 describes these results in detail.

4.2.2 Collecting Data

To train word embedding models that are security and privacy specific, we needed a large corpus containing the terms related to these domains used appropriately in sentences. To gather this corpus, we collected a large set of research papers in the security and privacy domain, as these papers use the sought after terms, thus providing a relevant context for training the word embeddings. To build this corpus of research papers, we used *arxiv.org* as the data source, one of the biggest repositories of publicly available research documents. It hosts papers from various areas, including Computer Science. We manually analyzed the available categories and identified *Cryptography and Security* as most relevant for our study and downloaded papers from this category to train our models. Specifically, we downloaded a total of 4972 research papers from *arxiv.org* in the Cryptography and Security category (CS.CR).

To automate the process of searching and downloading these papers, we developed a custom web crawler in Python using the *requests*[34] and *BeautifulSoup*[36] libraries. We searched *arxiv.org* with the text string *'CS.CR'* to get the URLs with the search parameters for the web pages that contain the links of the relevant paper documents. We then used the *requests* library to fetch the entire page source for the search results URL in HTML format.

Next, we used *BeautifulSoup* to consume this HTML and searched for specific HTML tags in the page source. As can be seen in Fig 4.2, the download links appear within a 'div' tag with class 'is-marginless' and the URL begins with 'https://arxiv.org/pdf/'.

Based on this observation, we identified the 'div' tags with the download links of the paper PDF files, which were then downloaded as shown:

```
import requests
from bs4 import BeautifulSoup
```

```
<div class="is-marginless">
<p class="list-title is-inline-block"><a href="https://arxiv.org/abs/2004.14861">arXiv:2004.14861</a>
<span> [<a href="https://arxiv.org/pdf/2004.14861">pdf</a>, <a href="https://arxiv.org/format/2004.14861">other
</a>] </span>
</p>
<div class="tags is-inline-block">
<span class="tag is-small search-hit tooltip is-tooltip-top" data-tooltip="Cryptography and Security">cs.CR</spa
n>
<span class="tag is-small is-grey tooltip is-tooltip-top" data-tooltip="Machine Learning">cs.LG</span>
<span class="tag is-small is-grey tooltip is-tooltip-top" data-tooltip="Machine Learning">stat.ML</span>
</div>
</div>
```

Figure 4.2: Sample 'arxiv.org' search result HTML source code

```
url = "https://arxiv.org/search/?searchtype=..."

htmlArxiv = requests.get(searchURL)

soup = BeautifulSoup(htmlArxiv.text, "html.parser")

listDiv = soup.findAll('div', {'class', "is-marginless"})

...

start = divItem.find("https://arxiv.org/pdf/")
```

The script moved between the identified pages by updating the URL query range parameters, until downloading the target number of papers.

To train the word embeddings using the downloaded data, *pyfasttext* requires a single file that contains the complete data in text format. To extract the text from the downloaded PDF files, we evaluated multiple tools and libraries such as *PDFMiner*[14], *PyPDF2*[31], *PyMuPDF*[35], and *Tika-Python*[11]. By comparing the quality of the text extracted by each library, we found that the performance of *Tika-Python* was the most suitable.

Tika-Python is a Python binding of Apache Tika, which is a content analysis toolkit written in Java. Then, using this library, we extracted the entire content of the downloaded PDFs. The extracted text includes additional information, such as page numbers, text from tables, equations, and symbols. To eliminate this noise from the extracted text, we pre-processed the extract text to remove all characters except alphabet, full stops (.) and commas (,).

This filtering step ensured that the model could learn word representations, without being swayed by the noise while preserving the usage context of the words. The resulting text was then fed to the *pyfasttext* library, which generated our custom word embeddings:

```
from pyfasttext import FastText
from tika import parser
import re
parsed = parser.from_file("pdf/" + str(i) + ".pdf")
try:
    text += parsed["content"] + "\n\n\n\n\n"
except:
    continue
regex = re.compile('[^a-zA-Z\s\n]')
textRE = regex.sub('', text)
model = FastText()
model.skipgram(input='fasttextInput.txt',
output='model', epoch=1, lr=0.7, wordNgrams=5)
```

4.2.3 Training Word Embeddings

As discussed previously, to train the word embeddings, we used the Python library *pyfasttext*. To ensure that the chosen word embedding model for classifying the relevant software requirements is the most suitable, we performed multiple experiments by training different models. To train multiple word embeddings, we altered the values of different hyperparameters, such as the number of epochs and the number of dimensions. We trained these models with 500, 700, 800, and 1000 epochs, with 50 and 100 dimensional word representations. We

then evaluated each of these models on the test set in order to determine the best model to be used for the recommendation system.

As described in Section 4.2.1, the pre-trained model performed best with the threshold value of 0.75. Thus, we evaluated each of the custom models with the same threshold. As discussed later in Section 5, the best models were trained with 700 and 1000 epochs, respectively, with the word representations of 100 dimensions. However, we found the recall values to be considerably higher in most cases, when the word representations are of 50 dimensions. To check whether the performance of these models varies with the threshold value, we also evaluated them with the threshold value of 0.8.

Based on our analysis, we found that the threshold of 0.8 significantly improves the results. The best model with the threshold of 0.8 had 50 dimensions and needed 800 epochs to train. However, having thoroughly examined the results, we noticed that many of the identified features contained 2 words. When compared with individual word tokens from the software requirements, the similarity score for these features would always be low. In a bid to further improve the performance of the models, we also calculated the similarity scores of the identified keywords with all possible bigrams (a set of two words occurring consecutively in a string) from the software requirements. After re-analyzing the results following the incorporation of the bigrams in our comparison, we discovered that the inclusion of bigrams improved the performance of all models. We finally observed that the model trained with 1000 epochs and 50 dimensions had the best performance across all the experimental setups. Section 5 describes the results in detail.

4.2.4 Gathering Test Data for Evaluation

To assess the effectiveness of our recommendation tool, we curated a set of software requirements from 10 open-source software projects that contained CPI and included a requirements specification document. We then manually inspected these requirements specifications for the presence of the keywords identified in Section 4.1. Our findings indicated that a subset of these keywords was not included in any of the inspected requirements. Hence, to ensure that all keywords would be represented in in the test data, we augmented the combined set of all software requirements with synthesized software requirements. To create the synthesized requirements for the excluded features, we reverse-engineered the requirements from the GitHub project repositories, which contained the features described in Section 4.1. We finally obtained a total of 553 software requirements (536 real and 17 synthesized) for the test dataset.

Having created this test dataset, we manually labelled each software requirement with either a 0 ('Not Relevant to TEE') or a 1 ('Relevant to TEE'). The test dataset contained 70 'Relevant to TEE' and 483 'Not Relevant to TEE' software requirements.

4.2.5 Test Setup

To compare the identified features, we first pre-processed each requirement to extract the most important keywords and optimize the comparison process as follows:

- Lowercase the requirement string to ensure that the results are unaffected by capitalization (Python treats uppercase and lowercase strings dissimilarly).
- Parse the requirement string into word tokens to be able compare all words in the requirement string individually.

- Remove all English language stopwords from the word tokens. Stopwords are commonly used auxiliaries, such as articles, prepositions, and helper verbs. These stopwords would not match the identified keywords, so retaining them would incur an unnecessary processing overhead on the system.

- Remove the word *user* from the word tokens, as software requirements are often framed with the word 'user' as the actor. This word would not match any of the identified keywords, causing an unnecessary overhead.

Finally, we used these pre-processed requirements for comparison.

To compare the pre-processed requirements with the identified features, we identified the words and bigrams in the software requirements most similar to the set of the identified features. This comparison was performed using a similarity score provided by *pyfasttext*. Intuitively, if a requirement contains a word very similar to one of the identified keywords, it is highly probable that this requirement corresponds to a security or privacy sensitive feature, whose implementation should be protected in TEE. Guided by this intuition, we calculated the similarity score for each of the word tokens and bigrams in the software requirements with each of the identified keywords. If any of the similarity scores was higher than the threshold, we classified the corresponding software requirement as 'Relevant to TEEs'

4.2.6 Evaluation Metrics

To evaluate the results obtained by classifying the software requirements in the test dataset, we used some of the standard techniques for evaluating classification performance: accuracy, precision, recall, and F-1 score. Among these metrics, precision, recall, and F-1 score are often used for skewed datasets.

Chapter 5

Results and Discussions

In this section, we examine the results obtained from evaluating the analysis of GitHub repositories and the automated annotation of software requirements, described in Section 4.

5.1 GitHub Repository Analysis

Table 5.1 presents high level results of the analysis of the GitHub repositories. Recall that we pre-selected those repositories in which at least one source file included *sgx_urts.h*. Out the resulting 429 repositories, 85 repositories had no code running in TEE, while the remaining 344 repositories had some code isolated in TEE. These remaining repositories were selected for further analysis.

Table 5.1: Repository Categorization

Category	Number of Repositories
Relevant	133
Not Relevant	129
Needs Refactoring	82
N/A	85
Total	429

Out of the total 344 repositories, only 133 of them were found to use TEE properly, isolating only CPI, related to security or privacy. These projects were labelled as 'Relevant'. The remaining 211 repositories were found to have failed to use TEE optimally, and were labelled

as either 'Not Relevant' or 'Needs Refactoring'.

To understand how TEE is used in real projects and which program features are typically isolated, we further analyzed the 133 'Relevant' repositories. We manually looked through their enclave code, extracting TEE-isolated features, having identified 65 such features in total.

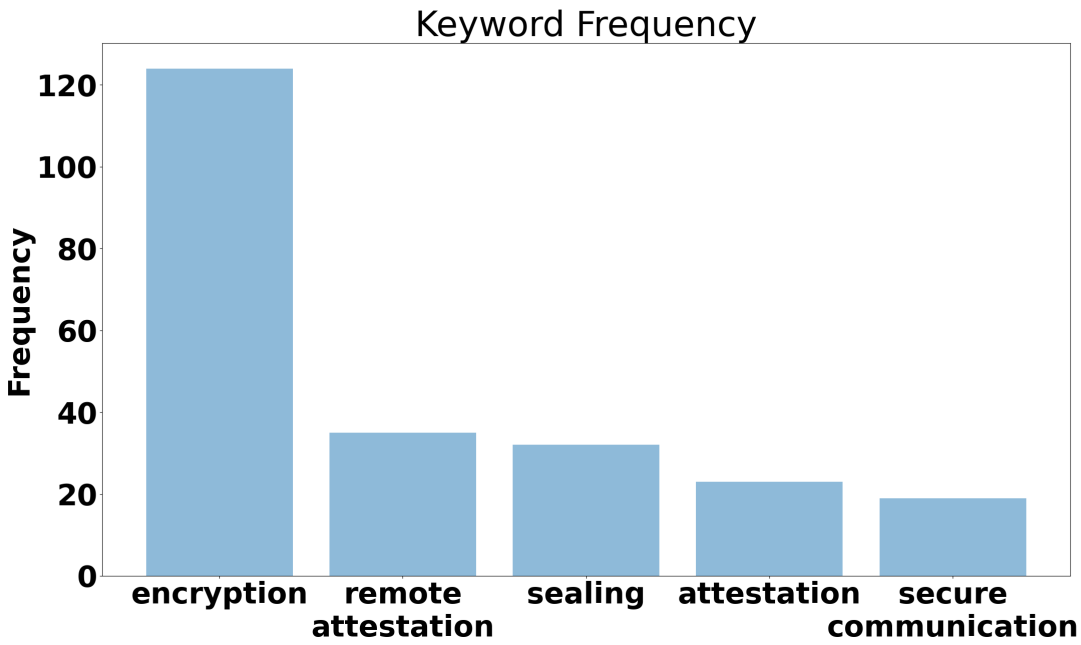

Figure 5.1: Top 5 TEE features as based on their frequency

Fig 5.1 depicts the top 5 most frequent features along with their frequencies. As per this figure, the feature 'encryption' has by far the highest frequency. The other 4 features 'remote attestation', 'sealing', 'attestation', and 'secure communication' have approximately similar frequencies. The remaining 60 features occurred only infrequently (i.e., between 2 and 10 times across all repositories).

Guided by these findings, we created our recommendation tool to classify the software requirements as part of the Requirements Engineering phase. Our goal is to assist developers

in identifying early on CPI-related features to be protected in TEE and also reducing the TCB size by keeping non-CPI features out of TEE.

5.2 Automated Annotation of Software Requirements

We next describe the results of classifying the software requirements with the *fastText* model, pre-trained on the English Wikipedia corpus. Then, we describe the results obtained using our context specific models, trained on research papers from the Cryptography and Security category in *arxiv.org*.

5.2.1 Pre-Trained fastText Model

Recall that we categorize the test dataset of software requirements into 'Relevant to TEE' or 'Not Relevant to TEE' categories. Due to the large size of this model's corpus, we inspected only the nearest neighbor words to our keywords, evaluating this model's quality. With 'encrypt' identified as the most frequent feature, we used this word to initially inspect the feasibility of using this model.

```
[('encrypting', 0.8346764445304871),
 ('encrypted', 0.8077129125595093),
 ('decrypt', 0.787101686000824),
 ('re-encrypt', 0.7765631675720215),
 ('encrypts', 0.7748447060585022),
 ('encryption', 0.7747678160667419),
 ('unencrypt', 0.7715140581130981),
 ('re-encrypts', 0.7087466716766357),
 ('Encrypting', 0.6968006491661072),
 ('encypted', 0.6933922171592712)]
```

Figure 5.2: Nearest Neighbours of word 'encrypt' using pre-trained model

Fig 5.2 shows that the nearest neighbours of 'encrypt' are variants of this term with a high

degree of similarity.

Encouraged by promising results, we continued using this pre-trained model and evaluated it on a test set of open-source software requirements. We performed various experiments by varying the similarity score threshold between 0.65-0.8.

Table 5.2 shows the results obtained for this model.

Table 5.2: Classification evaluation from pre-trained model

Threshold	Accuracy	Precision	Recall	F-1
0.65	87.3	50	45.7	47.8
0.7	87.2	49.2	44.3	46.6
0.75	**89.3**	**61.7**	**41.4**	**49.6**
0.8	89.9	68.4	37.1	48.1

We found that the pre-trained model classified the software requirements with a high degree of accuracy. However, as described in Section 4.2.4, the test dataset consisted of 70 'Relevant to TEE' and 483 'Not Relevant to TEE' software requirements. As this test data is highly skewed in nature, accuracy is not the most suitable measure for evaluating the tool's performance. As an illustration, if the tool were to classify each software requirement as 0 'Not Relevant to TEE', the accuracy score would still be 90.1%. Thus, to evaluate the model's performance, we applied other evaluation metrics—precision, recall, and F-1 score—which are known to accurately account for the skew in datasets. Hence, we report these metrics for each of the experiments performed. Additionally, we show the a demonstration of the tool on a small subset of the test software requirements, using the experimental setup with the best evaluation metrics. We randomly chose this subset by selecting 6 requirements pre-verified as having no CPI-related features and 4 requirements with identified CPI-related features that would benefit from TEE protection.

The results in Table 5.2 shows that recall and F-1 scores are low across all thresholds, while precision is comparatively higher for higher threshold values. These evaluations are

Table 5.3: Classification results with pre-trained model at threshold 0.75

Software Requirement	Actual TEE Relevance	Predicted TEE Relevance
The system shall display links to destination details, hotels, things to do and cars on the top of the web page	False	False
The user of the system shall be authenticated with a username and password	True	False
Password shall consist of minimum 8 and maximum 16 characters of which at least 2 must be uppercase letters, at least 2 must be lowercase letters, at least 2 must be numbers, and at least 2 must be special characters	False	False
Roommate shall be able to add any expense incurred.	False	False
The system shall verify the information, save the order as pending, and forward payment information to the accounting system.	True	True
The user shall click My Account to view account profile	False	False
The system shall filter the review to make sure the review does not contain inappropriate content.	False	False
The password will be saved in an encrypted format in the database for security purposes.	True	True
The system shall impose age restrictions while viewing booking movies with corresponding ratings certifications.	False	True
The application configurations shall be loaded from the configuration files located in the /WEB-INF directory.	True	False

corroborated in Table 5.3, where we see that the model is only able to correctly categorize only 2 of the 4 software requirements that are CPI-related. Recall is the ratio of the number of requirements the tool classifies as 'Relevant to TEE' vs. the actual number of requirements that contain CPI and should be protected in TEE. As a recommendation system, the tool should be designed to maximize the recall value, thus rendering this pre-trained model unsuitable. Indeed, this model lacks context sensitivity with respect to the security and privacy domain, having been trained on all-encompassing topics of the Wikipedia English articles.

To improve the results, we then experimented with custom word embeddings models, whose training content is more context specific to security and privacy.

5.2.2 Custom Trained Models

All custom word embeddings were trained on the same dataset extracted from the research papers downloaded from *arxiv.org*. The tested models were generated by varying the number of epochs and the number of dimensions. Each model was trained and saved for further analysis. In the end, the most suitable model was identified.

As shown in Table 5.2, the pre-trained model's best performance is achieved with the similarity score threshold of 0.75.

Thus, we tested the custom models with the same threshold. Table 5.4 presents the classification analysis results for each of these models.

Table 5.4: Classification evaluation from trained models at threshold 0.75

Number of Epochs	Dimensions	Accuracy	Precision	Recall	F-1
500	100	90.4	60.8	68.6	64.4
	50	87.5	50.5	67.1	57.7
700	**100**	**90.9**	**63.5**	**67.1**	**65.3**
	50	83.3	41.4	75.7	53.5
800	100	90.8	62.7	67.1	64.8
	50	57.5	21.1	85.7	33.8
1000	**100**	**90.9**	**63.5**	**67.1**	**65.3**
	50	88.8	54.1	75.7	63.1

To identify the best performing model, we used F-1 score, as it incorporates both precision and recall. Based on the F-1 scores, we discovered that the models generated with either 700 or 1000 epochs and 100 dimensions yielded the F-1 scores of 65.3. For the threshold of 0.75, both of the best performing models were trained with 100 dimensional word representations. Table 5.5 shows the outcome of the classification on the subset of the test requirements. We see that this subset has been classified fairly well with just 1 false positive and false negative respectively.

Table 5.5: Classification results from model trained with 700 epochs, 100 dimensions, and threshold 0.75

Software Requirement	Actual TEE Relevance	Predicted TEE Relevance
The system shall display links to destination details, hotels, things to do and cars on the top of the web page	False	False
The user of the system shall be authenticated with a username and password	True	True
Password shall consist of minimum 8 and maximum 16 characters of which at least 2 must be uppercase letters, at least 2 must be lowercase letters, at least 2 must be numbers, and at least 2 must be special characters	False	True
Roommate shall be able to add any expense incurred.	False	False
The system shall verify the information, save the order as pending, and forward payment information to the accounting system.	True	True
The user shall click My Account to view account profile	False	False
The system shall filter the review to make sure the review does not contain inappropriate content.	False	False
The password will be saved in an encrypted format in the database for security purposes.	True	True
The system shall impose age restrictions while viewing booking movies with corresponding ratings certifications.	False	False
The application configurations shall be loaded from the configuration files located in the /WEB-INF directory.	True	False

However, by analyzing the recall values across most of the experimental setups, we discovered that 50 dimensional word representations yielded higher higher recall scores and considerably lower precision scores, as compared to 100 dimensional models. Aiming to maximize the recall score, as the most important criteria for a recommendation system[9], we then experimented with the similarity threshold of 0.8. We hypothesized that the threshold of 0.8 would increase the model effectiveness against false positives, thus further increasing precision and F1-score.

Table 5.6 presents the scores obtained for each experimental setup with the threshold of 0.8.

Table 5.6: Classification evaluation from trained models at threshold 0.8

Number of Epochs	Dimensions	Accuracy	Precision	Recall	F-1
500	100	90.4	69.8	42.9	53.1
	50	89.3	60	47.1	52.8
700	100	90.4	69.8	42.9	53.1
	50	90.8	62.3	68.6	65.3
800	100	89.9	67.5	38.6	49.1
	50	**90.9**	**63.2**	**68.6**	**65.8**
1000	100	89.9	67.5	38.6	49.1
	50	90.8	62.3	68.6	65.3

The model trained with 800 epochs and 50 dimensions is best performing. As per our hypothesis, the threshold of 0.8 indeed has a tighter bound, so a smaller number of non-relevant requirements end up marked as 'Relevant to TEE', increasing the precision scores.

Table 5.7: Classification results from model trained with 800 epochs, 50 dimensions, and threshold 0.8

Software Requirement	Actual TEE Relevance	Predicted TEE Relevance
The system shall display links to destination details, hotels, things to do and cars on the top of the web page	False	False
The user of the system shall be authenticated with a username and password	True	True
Password shall consist of minimum 8 and maximum 16 characters of which at least 2 must be uppercase letters, at least 2 must be lowercase letters, at least 2 must be numbers, and at least 2 must be special characters	False	True
Roommate shall be able to add any expense incurred.	False	False
The system shall verify the information, save the order as pending, and forward payment information to the accounting system.	True	True
The user shall click My Account to view account profile	False	False
The system shall filter the review to make sure the review does not contain inappropriate content.	False	False
The password will be saved in an encrypted format in the database for security purposes.	True	True
The system shall impose age restrictions while viewing booking movies with corresponding ratings certifications.	False	False
The application configurations shall be loaded from the configuration files located in the /WEB-INF directory.	True	False

As can be seen Table 5.7, that this subset has been classified fairly well with just 1 false positive and false negative respectively.

We then visualized the word representations of the models generated with 800 epochs and 50 or 100 dimensional word representations. We chose to visualise the top–25 nearest neighbors of the word 'encrypt' as it is the most frequent feature. The visualisations in Fig 5.3 and Fig 5.4 show that the words closest to 'encrypt' are semantically more similar in 50 dimensions, thus making it possible to keep the threshold of 0.8, which increases the precision score.

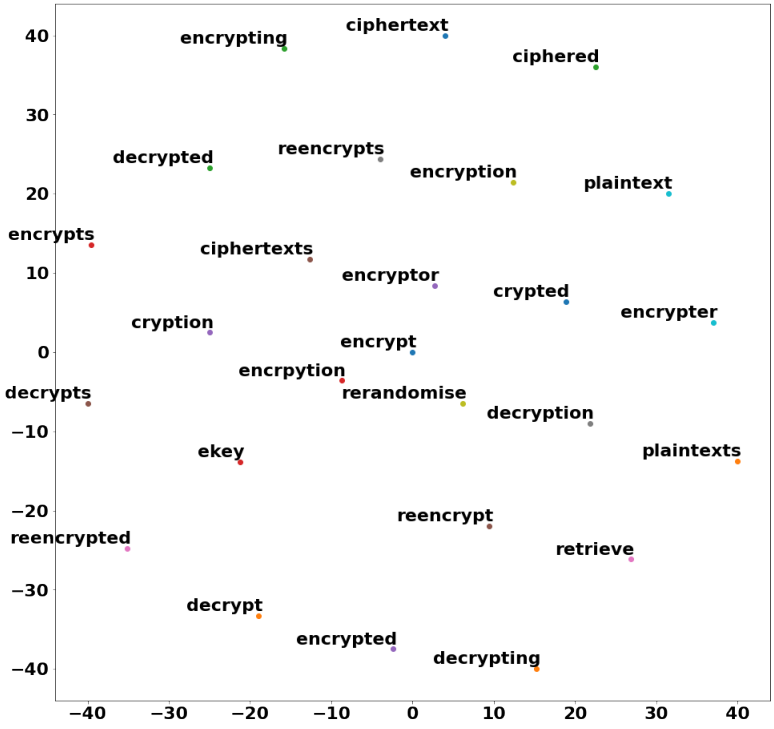

Figure 5.3: Top 25 Nearest Neighbours of 'encrypt' in 50 dimensions

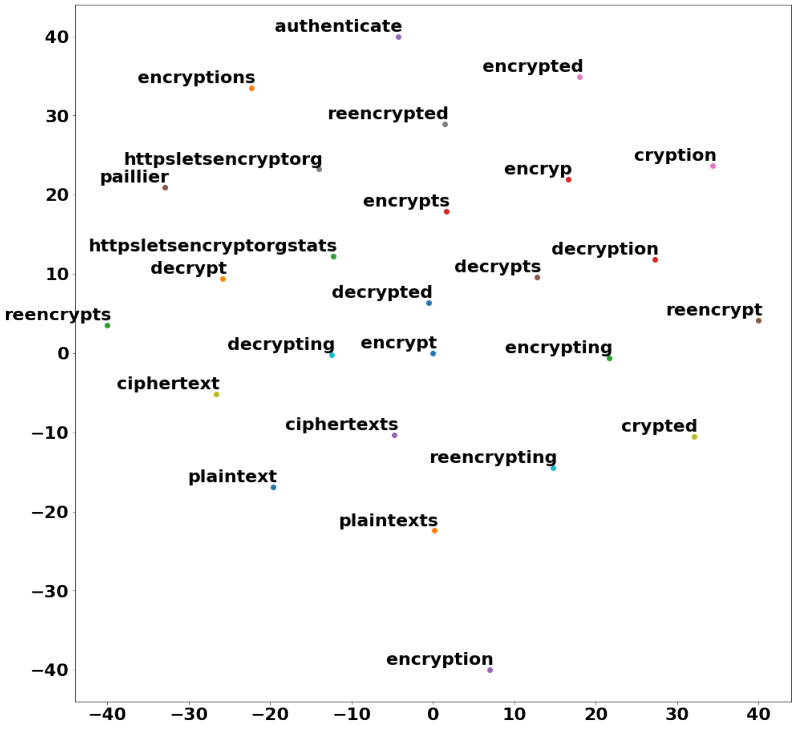

Figure 5.4: Top 25 Nearest Neighbours of ”encrypt” in 100 dimensions

To further improve our recall score, we then compared the identified keywords with all possible bigrams from the requirement strings. We observed a significant improvement in the recall values after the inclusion of bigrams in the comparison, as evident in Table 5.8, at the expense of increasing the number of false positives.

Table 5.8: Classification evaluation from trained models including bigrams at threshold 0.8

Number of Epochs	**Dimensions**	**Accuracy**	**Precision**	**Recall**	**F-1**
500	100	89.5	57.3	67.1	61.8
	50	74.9	31.3	82.8	45.5
700	100	90.4	62	62.9	62.4
	50	75.4	33	91.4	48.5
800	**100**	**88.4**	**53.3**	**68.6**	**60**
	50	72	29.9	90	44.8
1000	100	88.8	54.8	65.7	59.7
	50	77.4	34.8	90	50.2

Table 5.9: Classification results from model trained with 800 epochs, 50 dimensions, and threshold 0.8 after including bigrams

Software Requirement	**Actual TEE Relevance**	**Predicted TEE Relevance**
The system shall display links to destination details, hotels, things to do and cars on the top of the web page	False	False
The user of the system shall be authenticated with a username and password	True	True
Password shall consist of minimum 8 and maximum 16 characters of which at least 2 must be uppercase letters, at least 2 must be lowercase letters, at least 2 must be numbers, and at least 2 must be special characters	False	True
Roommate shall be able to add any expense incurred.	False	False
The system shall verify the information, save the order as pending, and forward payment information to the accounting system.	True	True
The user shall click My Account to view account profile	False	True
The system shall filter the review to make sure the review does not contain inappropriate content.	False	False
The password will be saved in an encrypted format in the database for security purposes.	True	True
The system shall impose age restrictions while viewing booking movies with corresponding ratings certifications.	False	False
The application configurations shall be loaded from the configuration files located in the /WEB-INF directory.	True	False

Consequently, the precision values and F-1 scores decreased considerably for models with 50 dimensions, where the recall was higher. The same observation can be seen in the classification demonstration on the test subset in Table 5.9, where we see a higher number of false positives.

To limit the number of false positives, we increased the threshold for comparison with bigrams to 0.85. Table 5.10 shows the scores obtained after incorporating the bigrams with a higher threshold.

Table 5.10: Classification evaluation from trained models at 1–gram threshold 0.8 and bigram threshold 0.85

Number of Epochs	Dimensions	Accuracy	Precision	Recall	F-1
500	100	91.1	67.2	58.6	62.6
	50	89.7	58	67.1	62.2
700	100	91.3	68.3	58.6	63.1
	50	90.2	58.3	80	67.5
800	100	90.9	69.2	51.4	59
	50	90.1	57.7	80	67.1
1000	100	90.2	65.4	48.6	55.7
	50	**90.6**	**59.6**	**80**	**68.3**

The recall values improved in all cases, even though the precision slightly deteriorated when compared to the scores in Table 5.6. Since the overall F-1 scores improved as well, we concluded that the best model is the one generated with 1000 epochs and 50 dimensions. Table 5.11 demonstrates the application of our recommendation tool on a representative subset of the test dataset. Largely, the predicted TEE relevance matches that of the real world TEE relevance with 2 exceptions, representing false positives and negatives. As can be seen from Table 5.12, based on the predictions made by the best model, for this test data, the tool reported 41 false positives. In other words, 41 requirements were erroneously recommended for TEE protection, yielding comparatively lower precision.

Table 5.11: Classification results from model trained with 1000 epochs, 50 dimensions, and thresholds 0.8 and 0.85 for 1–grams and bigrams

Software Requirement	**Actual TEE Relevance**	**Predicted TEE Relevance**
The system shall display links to destination details, hotels, things to do and cars on the top of the web page	False	False
The user of the system shall be authenticated with a username and password	True	True
Password shall consist of minimum 8 and maximum 16 characters of which at least 2 must be uppercase letters, at least 2 must be lowercase letters, at least 2 must be numbers, and at least 2 must be special characters	False	True
Roommate shall be able to add any expense incurred.	False	False
The system shall verify the information, save the order as pending, and forward payment information to the accounting system.	True	True
The user shall click My Account to view account profile	False	False
The system shall filter the review to make sure the review does not contain inappropriate content.	False	False
The password will be saved in an encrypted format in the database for security purposes.	True	True
The system shall impose age restrictions while viewing booking movies with corresponding ratings certifications.	False	False
The application configurations shall be loaded from the configuration files located in the /WEB-INF directory.	True	False

Table 5.12: Confusion Matrix for model trained with 800 epochs, 50 dimensions and a 1–gram threshold of 0.8 and bigram threshold of 0.85

n = 553	Predicted No	Predicted Yes
Actual No	445	38
Actual Yes	14	56

Since the tool is a recommendation system, erroneously suggesting requirements for TEE protection is acceptable, as it is the developer who finally decides whether to follow these suggestions. Our aim was to maximize the recommendations for requirements that actually need TEE protection, as indicated by our higher recall score. Despite having a high recall score, a very low precision score would be undesirable, implying that more requirements than necessary are recommended for TEE protection. Hence, the best performing model was determined based on the F-1 scores.

Further inspection of the false positive cases showed that many erroneously recommended requirements contained the words 'password' and 'login'. However, these requirements were unrelated to the 'password' or 'login' features. As an example of a false positive software requirement, consider *The password shall be at least 8 characters long and must contain at least 1 number and 1 special character*. Although this requirement mentions the keyword 'password', it does not deal with the secure storage of passwords, neither does it deal with the usage of a password in the login process.

That is, it merely defines the rules that must be satisfied for passwords to be valid, a requirement that can be satisfied by performing field validation using string matching without any involvement of TEE. However, because the word 'password' matches our identified keyword list, it is marked as a false positive.

On the basis of the evaluation results and the demonstrations we conclude that the recommendation tool can help inform developers about CPI-related features needing TEE protection, as early as the Requirements Engineering phase.

Chapter 6

Threats to Validity

In this section we discuss a few internal and external factors that threaten the validity of the results of this study.

6.1 Internal Threats

The findings presented for the analysis of the GitHub repositories are based on the search results returned by the GitHub search API. If Intel were to change their implementations for writing code for SGX that no longer warrants the need for including *sgx_urts.h* in the source code, this approach would not be able to provide an exhaustive list of projects. Similarly, this study is also based on the premise that SGX code can be developed only on C/C++ platforms. Future enhancements to include additional coding platforms would also render the results presented incomplete. These findings are also likely to suffer from experimenter bias as the categorization was performed manually.

The classification results and model evaluations are based on the set of features identified from the analysis of GitHub repositories and the word embeddings trained on the research papers from the security and privacy domain. A potential threat to the validity of these results would be the introduction of new CPI-related features in the future, not presently represented either in the training dataset or in the list of identified keywords. Such a scenario would require the need to retrain the word embeddings with additional research documents

that contain the newly introduced features along with including these features in the list of identified keywords.

6.2 External Threats

The identification of features that optimally utilize TEE protection is based upon recommendations by Intel and also derived only from SGX based projects. Additionally, we only focused on GitHub as the source for open source projects that use Intel SGX. It is likely that inclusion of other TEE hardware or code hosting platforms could lead to discovery of previously unidentified features that may benefit from TEE protection.

Chapter 7

Limitations and Future Work

Despite our objectives to develop the best possible recommendation tool with the available datasets and other information, our approach does fall short in some aspects.

To classify the software requirements, our approach relies on keyword comparison. Hence, our approach's effectiveness hinges on having detailed software requirements, so as to ensure that they contain important keywords for use in the comparison algorithm. However, inadequately phrased requirements may lack important terms, causing our approach to miss-classify them as 'Not Relevant to TEE'.

The test dataset is highly skewed, due to the high disparity between the number of 'Relevant to TEE' and 'Not Relevant to TEE' requirements. However, considering the non-artificial nature of the software requirements in the test dataset, with its fairly low percentage of CPI-related requirements, we consider our study's result as representative of the real-world TEE-based projects.

Our best model has the precision score of 59.6 to accommodate those requirements that contain keywords as 'password' or 'login' but are unrelated to security or privacy. This fairly low precision score is due to a slightly larger number of such false positives. The corresponding requirements would be erroneously recommended for as needing TEE protection.

Additionally, the list of keywords was statically identified by manually analyzing the GitHub repositories. This list is highly dependent on the inputs and best practices recommended by

Intel and other research articles. We attempted to overcome this limitation by experimenting with similarity measures for the keywords and varying threshold values. However, the trained models are still dependent on the training text that may not encompass all security and privacy related keywords.

To overcome some of these limitations, one could train a machine learning classification model on a large set of software requirements. The resulting model would then be able to classify new requirements without having to match keywords. However, in the absence of a readily available large set of software requirements, especially those 'Relevant to TEE', we could not explore this approach. This approach would have addressed the aforementioned limitation of using the static list of keywords, as it would eliminate keyword matching and instead classify based on the features learned during training. Similarly, this classification model could also reduce the number of false positives by making decisions based on all words in the requirement strings rather than only word tokens or bigrams.

Chapter 8

Conclusions

In this paper, we presented a novel approach that automatically annotates the software requirements to incorporate Trusted Execution Environments in the Requirements Engineering phase of the software development life cycle. By introducing TEE early in the development cycle, this work aims to reduce the need for refactoring source code to take advantage of TEE.

Our analysis of 429 GitHub repositories that used the Intel SGX hardware revealed that the majority of them used TEE sub-optimally. Motivated by these findings, we put forward an approach that introduces the concepts of CPI in need of TEE protection from the project's inception.

We concretely realized our approach as a recommendation tool that classifies software requirements as 'Relevant to TEE' or 'Not Relevant to TEE'. The word embeddings were trained on the content of 4972 computer security/privacy research papers. Our work confirms that comparing the software requirements with the identified keywords based on the trained word embeddings yields promising results: NLP techniques can aid developers in identifying features that should be isolated in TEE early in the development cycle. Pushing TEE awareness closer to the start of the software development process can eliminate expensive and error-prone refactoring of completed implementation, ensuring proper TEE protection of security and privacy related features.

Bibliography

[1] [n.d.]. Adventures of an Enclave (SGX / TEEs). *By Leland Lee* ([n. d.]). https://hackernoon.com/adventures-of-an-enclave-sgx-tees-9e7f8a975b0b

[2] [n.d.]. Core. *OP*. https://optee.readthedocs.io/en/latest/architecture/core.html#smc

[3] [n.d.]. Open Portable Trusted Execution Environment. *OP* ([n. d.]). https://www.op-tee.org/

[4] [n.d.]. Secure Enclave overview. *Apple Support* ([n. d.]). https://support.apple.com/guide/security/secure-enclave-overview-sec59b0b31ff/web

[5] 2016. Word2Vec Tutorial - The Skip-Gram Model. *Word2Vec Tutorial - The Skip-Gram Model · Chris McCormick* (Apr 2016). http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

[6] 2020. *GitHub Search API*. Retrieved Feb 9, 2020 from https://developer.github.com/v3/search/#search-repositories

[7] Admin. 2019. Intel® Software Guard Extensions (Intel® SGX) Web-Based Training. *Intel® Software* (Aug 2019). https://software.intel.com/en-us/articles/intel-sgx-web-based-training

[8] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics* 5 (2017), 135–146.

[9] Kirill Bondarenko. 2019. Precision and recall in recommender systems. And some metrics stuff. *Medium* (Feb 2019). https://medium.com/@bond.kirill.alexandrovich/precision-and-recall-in-recommender-systems-and-some-metrics-stuff-ca2ad385c5f8

[10] Carole Cadwalladr and Emma Graham-Harrison. 2018. Revealed: 50 million Facebook profiles harvested for Cambridge Analytica in major data breach. *The guardian* 17 (2018), 22.

[11] Chrismattmann. 2020. chrismattmann/tika-python. *GitHub* (Mar 2020). https://github.com/chrismattmann/tika-python

[12] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* 2016, 086 (2016), 1–118.

[13] Sanket Doshi. 2019. Skip-Gram: NLP context words prediction algorithm. https://towardsdatascience.com/skip-gram-nlp-context-words-prediction-algorithm-5bbf34f84e0c.

[14] Euske. 2020. euske/pdfminer. *GitHub* (Jan 2020). https://github.com/euske/pdfminer

[15] Facebookresearch. 2020. facebookresearch/fastText. *GitHub* (Feb 2020). https://github.com/facebookresearch/fastText/

[16] Henk WM Gazendam. 2004. Organizational Semiotics: a state of the art report. *Semiotix* 1, 1 (2004), 1–5.

[17] Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. 2018. Learning Word Vectors for 157 Languages. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

[18] Barbara Hauer. 2015. Data and information leakage prevention within the scope of information security. *IEEE Access* 3 (2015), 2554–2565.

[19] Alice Hutchings, Russell G Smith, Lachlan James, et al. 2013. Cloud computing for small business: Criminal and security threats and prevention measures. *Trends and issues in Crime and Criminal Justice* 456 (2013), 1.

[20] Johnm. 2019. Intel® Software Guard Extensions Tutorial Series: Part 1, Intel® SGX Foundation. *Intel® Software* (Oct 2019). https://software.intel.com/en-us/articles/intel-software-guard-extensions-tutorial-part-1-foundation

[21] Donald E Knuth, James H Morris, Jr, and Vaughan R Pratt. 1977. Fast pattern matching in strings. *SIAM journal on computing* 6, 2 (1977), 323–350.

[22] Krishan. 2017. Continuous Bag of Words (CBOW). *From Data to Decisions* (Jul 2017). https://iksinc.online/tag/continuous-bag-of-words-cbow/

[23] Simon Liu and Rick Kuhn. 2010. Data loss prevention. *IT professional* 12, 2 (2010), 10–13.

[24] Yin Liu, Kijin An, and Eli Tilevich. 2018. RT-trust: automated refactoring for trusted execution under real-time constraints. In *Proceedings of the 17th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences*. 175–187.

[25] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.

[26] Acorn Risc Machine. 1990. Family Data Manual. (1990).

[27] Gary McGraw. 2005. The security lifecycle-the 7 touchpoints of secure software-just as you can't test quality into software, you can't bolt security features onto code and expect it to become hack-proof security. *Software Development* 13, 9 (2005), 42–43.

[28] Mhoekstr. 2019. Intel® SGX for Dummies (Intel® SGX Design Objectives). *Intel® Software* (Oct 2019). https://software.intel.com/en-us/blogs/2013/09/26/protecting-application-secrets-with-intel-sgx

[29] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[30] Daniel Miller. 2011. Facebook hackers attempting to crack 600,000 accounts every day. *Daily Mail Online* (Oct 2011). https://www.dailymail.co.uk/sciencetech/article-2054994/Facebook-hackers-attempting-crack-600-000-accounts-day.html

[31] mstamy2. 2018. mstamy2/PyPDF2. *GitHub* (Jun 2018). https://github.com/mstamy2/PyPDF2

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[33] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. 1532–1543. Retrieved December 11, 2019 from http://www.aclweb.org/anthology/D14-1162

[34] Psf. 2020. psf/requests. *GitHub* (Feb 2020). https://github.com/psf/requests

[35] Pymupdf. 2020. pymupdf/PyMuPDF. *GitHub* (Mar 2020). https://github.com/pymupdf/PyMuPDF

[36] Leonard Richardson. [n.d.]. Beautiful Soup. *Beautiful Soup: We called him Tortoise because he taught us.* ([n. d.]). https://www.crummy.com/software/BeautifulSoup/

[37] Roskakori. [n.d.]. roskakori/pygount. *GitHub*. https://github.com/roskakori/pygount

[38] Rscosta. 2017. rscosta/SGXCryptoFile. *GitHub* (Aug 2017). https://github.com/rscosta/SGXCryptoFile

[39] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. 2015. Trusted execution environment: what it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/ISPA*, Vol. 1. IEEE, 57–64.

[40] Sarkar. 2019. A hands-on intuitive approach to Deep Learning Methods for Text Data - Word2Vec, GloVe and FastText. *Medium*. https://towardsdatascience.com/understanding-feature-engineering-part-4-deep-learning-methods-for-text-data-96c443

[41] Richa Sharma. 2017. Grounding Functional Requirements Classification in Organizational Semiotics. *Interdisciplinary Approaches to Semiotics* (2017), 151.

[42] Richa Sharma and Kanad K Biswas. 2015. Functional requirements categorization grounded theory approach. In *2015 International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*. IEEE, 301–307.

[43] Nishan Subedi. 2018. FastText: Under the Hood. *Medium* (Jul 2018). https://towardsdatascience.com/fasttext-under-the-hood-11efc57b2b3

[44] Vrasneur. 2018. vrasneur/pyfasttext. *GitHub* (Dec 2018). https://github.com/vrasneur/pyfasttext

[45] Nico Weichbrodt, Pierre-Louis Aublin, and Rüdiger Kapitza. 2018. sgx-perf: A performance analysis tool for Intel SGX enclaves. In *Proceedings of the 19th International Middleware Conference*. 201–213.

Appendices

Appendix A

Sample SGX based project

Here we describe a open source Cryptography application [38] that uses Intel SGX for securing its encryption operations.

filepath–Project_Root/SGXCryptoFile/CryptoEnclave/CryptoEnclave.edl

```
enclave {
    trusted {
        /*
        * This function decrypts a message.
        * @param encMessageIn
        *    the encrypted message containing MAC + IV + encrypted message.
        * @param len
        *    the length of the encMessageIn.
        * @param decMessageOut
        *    the destination of the decrypted message.
        * @param lenOut
        *    the length of the decMessageOut.
        */
        public void sgxDecryptFile([in,size=len] unsigned char *encMessageIn,
        size_t len, [out,size=lenOut] unsigned char *decMessageOut, size_t lenOut);
```

```
        /*
        * This function encrypts a message.
        * @param decMessageIn
        *    the original message
        * @param len
        *    the length of the decMessageIn.
        * @param encMessageOut
        *    the destination of the encrypted message
        *    containing MAC + IV + encrypted message.
        * @param lenOut
        *    the length of the encMessageOut.
        */
        public void sgxEncryptFile([in,size=len] unsigned char *decMessageIn,
        size_t len, [out,size=lenOut] unsigned char *encMessageOut, size_t lenOut);
    };

    untrusted {
        [cdecl] void printDebug([string,in] const char *str);
    };
};
```

This *Enclave Definition File* is the interface between the trusted and the untrusted code of the application. As seen in the code, the methods for encryption and decryption are placed in the TEE and the application can use these function calls to invoke the trusted methods and switch the execution context to the TEE. Similarly, the *printDebug* method is used by

the TEE to switch the context back to the operating system and continue the processing of the unsecure code.

filepath–Project_Root/SGXCryptoFile/CryptoEnclave/CryptoEnclave.cpp

```
#include "CryptoEnclave_t.h"

#include "sgx_trts.h"
#include "sgx_tcrypto.h"
#include "stdlib.h"
#include <string.h>

static sgx_aes_gcm_128bit_key_t key = { 0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7,
                                        0x8, 0x9, 0xa, 0xb, 0xc, 0xd, 0xe, 0xf };

void sgxDecryptFile(unsigned char *encMessageIn, size_t len,
                unsigned char *decMessageOut, size_t lenOut)
{
    uint8_t *encMessage = (uint8_t *) encMessageIn;
    uint8_t p_dst[lenOut];
        sgx_status_t ret;

        printDebug("INIT ENCLAVE DECRYPTION...");

    ret = sgx_rijndael128GCM_decrypt(
        &key,
        encMessage + SGX_AESGCM_MAC_SIZE + SGX_AESGCM_IV_SIZE,
```

```
        lenOut,
        p_dst,
        encMessage + SGX_AESGCM_MAC_SIZE, SGX_AESGCM_IV_SIZE,
        NULL, 0,
        (sgx_aes_gcm_128bit_tag_t *) encMessage);

    if(ret == SGX_SUCCESS) printDebug("DECRYPT RESULT: SGX_SUCCESS");
    if(ret == SGX_ERROR_INVALID_PARAMETER) printDebug("DECRYPT RESULT: \
                                        SGX_ERROR_INVALID_PARAMETER");
    if(ret == SGX_ERROR_OUT_OF_MEMORY) printDebug("DECRYPT RESULT: \
                                        SGX_ERROR_OUT_OF_MEMORY");
    if(ret == SGX_ERROR_UNEXPECTED) printDebug("DECRYPT RESULT: \
                                        SGX_ERROR_UNEXPECTED");

    memcpy(decMessageOut, p_dst, lenOut);
}

void sgxEncryptFile(unsigned char *decMessageIn, size_t len,
                unsigned char *encMessageOut, size_t lenOut)
{
    uint8_t *origMessage = (uint8_t *) decMessageIn;
    uint8_t p_dst[lenOut];
        sgx_status_t ret;

        printDebug("INIT ENCLAVE ENCRYPTION...");
```

```
// Generate the IV (nonce)
ret = sgx_read_rand(p_dst + SGX_AESGCM_MAC_SIZE, SGX_AESGCM_IV_SIZE);

if(ret == SGX_SUCCESS) printDebug("RAND RESULT: SGX_SUCCESS");
if(ret == SGX_ERROR_INVALID_PARAMETER) printDebug("RAND RESULT: \
                                    SGX_ERROR_INVALID_PARAMETER");
if(ret == SGX_ERROR_UNEXPECTED) printDebug("RAND RESULT: \
                                    SGX_ERROR_UNEXPECTED");

ret = sgx_rijndael128GCM_encrypt(
    &key,
    origMessage, len,
    p_dst + SGX_AESGCM_MAC_SIZE + SGX_AESGCM_IV_SIZE,
    p_dst + SGX_AESGCM_MAC_SIZE, SGX_AESGCM_IV_SIZE,
    NULL, 0,
    (sgx_aes_gcm_128bit_tag_t *) (p_dst));

if(ret == SGX_SUCCESS) printDebug("ENCRYPT RESULT: SGX_SUCCESS");
if(ret == SGX_ERROR_INVALID_PARAMETER) printDebug("ENCRYPT RESULT: \
                                    SGX_ERROR_INVALID_PARAMETER");
if(ret == SGX_ERROR_OUT_OF_MEMORY) printDebug("ENCRYPT RESULT: \
                                    SGX_ERROR_OUT_OF_MEMORY");
if(ret == SGX_ERROR_UNEXPECTED) printDebug("ENCRYPT RESULT: \
                                    SGX_ERROR_UNEXPECTED");

    memcpy(encMessageOut,p_dst,lenOut);
```

```
}
```

This file contains the definitions for the secure code that will be executed inside the TEE. This must contain the interface methods described in the edl file and can also contain definitions for any other methods that need secure execution.

filepath–Project_Root/SGXCryptoFile/CryptoFileApp/CryptoFileApp.cpp

```
#include <string.h>

#include "sgx_urts.h"
#include "CryptoEnclave_u.h"

#include "getopt.h"
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "cpuidh.h" //benchmark

#define SGX_AESGCM_MAC_SIZE 16
#define SGX_AESGCM_IV_SIZE 12

#define VERSION   "1.2.170624"
#define AUTHOR   "Ricardo Costa"

#define MAX_FILE_NAME_SIZE  512
```

```
#define ENCLAVE_FILE "CryptoEnclave.signed.so"

int encryptFile(sgx_enclave_id_t eid, const char* input, const char* output)
{
    FILE *ifp = NULL;
    FILE *ofp = NULL;
    int nread = 0;

    if((ifp = fopen(input, "rb")) == NULL)
    {
      printf("[APP ENCRYPT] Input File %s not found!\n",input);
      return -1;
    }

    if((ofp = fopen(output, "wb")) == NULL)
    {
      printf("[APP ENCRYPT] Error while creating output File %s\n",output);
      fclose(ifp);
      return -1;
    }

    fseek(ifp, 0, SEEK_END);
    long fsize = ftell(ifp);
    fseek(ifp, 0, SEEK_SET);  //same as rewind(f);
    unsigned char *message = (unsigned char*)malloc(fsize + 1);
    size_t readRes = fread(message, fsize, 1, ifp);
```

```
    size_t encMessageLen = (SGX_AESGCM_MAC_SIZE + SGX_AESGCM_IV_SIZE + fsize);
    unsigned char *encMessage = (unsigned char *)
        malloc((encMessageLen+1)*sizeof(unsigned char));

    start_time();

    sgxEncryptFile(eid, message, fsize, encMessage, encMessageLen);

    end_time();

    encMessage[encMessageLen] = '\0';

    fwrite(encMessage, encMessageLen, 1, ofp);

    free(encMessage);
    free(message);

    fclose(ifp);
    fclose(ofp);

    printf("[APP ENCRYPT] Encryption file (%s) created!\n",output);
    printf("[APP ENCRYPT] Final encryption time: %6.6f seconds.\n", secs);
}

int decryptFile(sgx_enclave_id_t eid, const char* input, const char* output)
{
```

```
FILE *ifp = NULL;
FILE *ofp = NULL;
int nread = 0;

if((ifp = fopen(input, "rb")) == NULL)
{
  printf("[APP DECRYPT] Input File %s not found!\n",input);
  return -1;
}

if((ofp = fopen(output, "wb")) == NULL)
{
  printf("[APP DECRYPT] Error while creating output File %s\n",output);
  fclose(ifp);
  return -1;
}

fseek(ifp, 0, SEEK_END);
long fsize = ftell(ifp);
fseek(ifp, 0, SEEK_SET);  //same as rewind(f);
unsigned char *message = (unsigned char*)malloc(fsize + 1);
size_t readRes = fread(message, fsize, 1, ifp);
size_t decMessageLen = fsize - (SGX_AESGCM_MAC_SIZE + SGX_AESGCM_IV_SIZE);
unsigned char *decMessage = (unsigned char *)
    malloc((decMessageLen+1)*sizeof(unsigned char));
```

```
    start_time();

    sgxDecryptFile(eid,message,fsize,decMessage,decMessageLen);

    end_time();

    decMessage[decMessageLen] = '\0';

    fwrite(decMessage, decMessageLen, 1, ofp);

    free(decMessage);
    free(message);

    fclose(ifp);
    fclose(ofp);

    printf("[APP DECRYPT] Decryption file (%s) created!\n",output);
    printf("[APP DECRYPT] Final decryption time: %6.6f seconds.\n", secs);
}

void printDebug(const char *buf)

{
    printf("ENCLAVE: %s\n", buf);
}
```

```
void printAppUsage()
{
    printf("\nSgxCryptoFile - App for Encrypting and Decrypting Files using\
                        Intel SGX. Version:%s Author:%s\n", VERSION, AUTHOR);
    printf("Usage: sgxCryptoFile [OPTIONS] [FILE]\n\n");
    printf("Options:\n");
    printf(" -d\tdecryption mode enabled\n");
    printf(" -e\tencryption mode enabled\n");
    printf(" -i\tinput file\n");
    printf(" -o\toutput file\n");
    printf("Example (Encryption): sgxCryptoFile -e -i \
                    [INPUT_FILE] -o [OUTPUT_FILE]\n");
    printf("Example (Decryption): sgxCryptoFile -d -i \
                    [INPUT_FILE] -o [OUTPUT_FILE]\n\n");
}

int main(int argc, char *argv[])
{
    int option = 0;
    int mode = 0;
    char inFileName[MAX_FILE_NAME_SIZE];
    char outFileName[MAX_FILE_NAME_SIZE];

    // Specifying the expected options
    while ((option = getopt(argc, argv,"edi:o:")) != -1) {
        switch (option) {
```

```
      case 'e' :
                mode = 1; /*Encryption enabled */
              break;
           case 'd' :
                mode = 2; /*Decryption enabled */
             break;
           case 'i' :
          if(optarg == NULL)
            exit(EXIT_FAILURE);

            strncpy(inFileName, optarg, MAX_FILE_NAME_SIZE);
              break;
           case 'o' :
          if(optarg == NULL)
            exit(EXIT_FAILURE);

            strncpy(outFileName, optarg, MAX_FILE_NAME_SIZE);
              break;
       default: printAppUsage();
              exit(EXIT_FAILURE);
     }
}

//Check if it is invalid mode
if (mode == 0)
{
```

```
printAppUsage();
exit(EXIT_FAILURE);
}

 // Setup enclave
sgx_enclave_id_t eid;
sgx_status_t ret;
sgx_launch_token_t token = { 0 };
int token_updated = 0;

//Init enclave
ret = sgx_create_enclave(ENCLAVE_FILE, SGX_DEBUG_FLAG,
                    &token, &token_updated, &eid, NULL);

if (ret != SGX_SUCCESS)
{
printf("sgx_create_enclave failed: %#x\n", ret);
exit(EXIT_FAILURE);
}

if(mode ==1)
{
//Encrypt a file
encryptFile(eid, inFileName, outFileName);
}
else
```

```
    {
    //Decrypt a file
    decryptFile(eid, inFileName, outFileName);
    }

    //Destroy Enclave
    sgx_destroy_enclave(eid);

    return 0;
}
```

This file contains the code for the actual executable that will be running when a user starts the application. This internally calls the secure methods through the interfaces described in the dl file. As can be seen, this includes the header *sgx_urts.h* which contains the method for enclave initialization. This causes this repository to be included in our search results from the GitHub search API.

Appendix B

Features using SGX protection

Fig B.1 shows the features that were identified during the GitHub repository analysis. As we can see, most features have small frequencies with the majority of them having a frequency of less that 3. The most used feature by far is encryption followed by remote attestation and sealing.

```
65
{'encryption': 124, 'remote attestation': 35, 'sealing': 32, 'attestation': 23, 'secure commun
ication': 19, 'secure data': 10, 'hashing': 10, 'signing': 10, 'verification': 9, 'train model
': 5, 'test model': 5, 'secure database': 5, 'secure messaging': 4, 'bitcoin implementation':
4, 'manage client requests': 4, 'manage ssl connection': 3, 'secure script execution': 3, 'man
age tls connection': 3, 'enforcement': 3, 'execution of contracts': 3, 'manage wallet': 3, 'lo
ad model': 2, 'predict': 2, 'machine learning calculations': 2, 'secure load': 2, 'sign': 2, '
verify': 2, 'password manager': 2, 'cluster calculation': 2, 'error calculation': 2, 'consensu
s algorithm': 2, 'secure firewall': 2, 'authentication': 2, 'manage zookeeper': 2, 'secure ses
sion': 2, 'wallet operations': 1, 'cryptography': 1, 'manage tls server': 1, 'secure flow tabl
es': 1, 'secure code execution': 1, 'predictions': 1, 'secure file encryption': 1, 'marshallin
g': 1, 'app initialization': 1, 'secure input - output': 1, 'secure connection': 1, 'secret pa
sscode generation': 1, 'query execution': 1, 'load sensitive content': 1, 'secure storage': 1,
 'manage encrypted data': 1, 'sigining': 1, 'keygen': 1, 'key generation': 1, 'secure evaluati
on': 1, 'verification of computational cycles': 1, 'secure creation': 1, 'manage auditor': 1,
'certification': 1, 'sgxssl api calls': 1, 'managing rule table': 1, 'key management': 1, 'han
dle client messages': 1, 'login': 1, 'manage payment': 1}
```

Figure B.1: Features that make use of Intel SGX along with their frequencies