UNIVERSITÉ
DE GENÈVE

FACULTY OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

MASTER'S THESIS

Confidential Computing of Machine Learning using Intel SGX.

Submitted By :

Prasad Koshy Jose

18341164

Supervisor :

Dr Eduardo Solana

Abstract

Prasad Koshy Jose

July 27, 2020

Deploying applications on the cloud requires a certain trust on third party infrastructure providers, privileged softwares and other tenants utilizing the same platform. Unfortunately, this trust has been violated multiple times in the past. The training phase of machine learning methods requires large scale processing of sensitive data sets owned by enterprises and customers. Outsourcing the training process to a cloud-based service poses significant security risks like breaches and compromised integrity of the data and the underlying model. An effective way to overcome this security issue is to employ a machine learning system which uses a hardware-assisted Trusted Execution Environment (TEE), Intel SGX, to run the computations on sensitive data sets. The risks are potentially reduced by delegating most of the trust to the hardware. Multi-user data sets are securely sealed using AES-GCM, a cryptographic Authenticated Encryption (AE) method and is transmitted to the machine learning code residing in a secure enclave, insusceptible to any vulnerabilities of a traditional system. This architecture is significantly secure with smaller attack surfaces and no significant performance overhead. This work also emphasizes on responsible software design and development which is key for minimizing risks of system penetration and unauthorised access.

Contents

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Dr Eduardo Solana, for his patient guidance, enthusiastic encouragement and useful critiques of this work and providing me with different ideas and perspectives to complete this thesis. I would also like to thank everyone who has supported me directly and indirectly to complete this thesis.

List of Figures

Acronyms

AE Authenticated Encryption. 1, 32

API Application Program Interface. 12

CNN Convolutional Neural Network. 19

CPU Central Processing Unit. 11

FHE Fully Homomorphic Encryption. 14

FLD Fisher's Linear Discriminant. 25

IV Initialization Vector. 39

LDA Linear Discriminant Analysis. 15

LMC Linear Means Classifier. 25

LOC Lines Of Code. 9

NN Neural Network(s). 11

OS Operating System. 11

RNN Recurrent Neural Network. 21

SGX Software Guard Extensions. 10

TCB Trusted Code Base. 10

TEE Trusted Execution Environment. 1, 10

TPM Trusted Platform Modules. 10

VM Virtual Machines. 27

Chapter 1

1 Introduction

The need for data security is always on the rise. With the rise of cloud technologies, security and privacy are the most significant concerns for both enterprises and consumers. Additionally, cloud computing piles on the stress as private data is stored and processed in multi-tenant hardware providers.

Traditional systems perform access control by separating software roles into multiple user modes. Privileged system software, like hyper-visors and operating systems, control and manage machine resources and other vital components of the system, the user developed applications rely not only on the correctness of the system software, i.e. that they are free of exploitable bugs, but also on the honesty of system softwares. The correctness of the system software is vastly exploited by hackers since this is very hard to verify. The main reason for this difficulty is that a software system comprises of millions of Lines Of Code (LOC) and very complex control and data flows. Most of the operating systems, including Windows and Linux, have a lot of legacy codes, which makes them susceptible to various malicious attacks. The analysis done by [1] shows that more than 10000 computer systems vulnerabilities are discovered each year. It is not a far fetched assumption that these vulnerabilities still linger in cloud technologies and that an adversary would exploit them to gain access to confidential processes and data.

The situation gets trickier when it comes to 'honesty'. With the emergence of cloud computing, many applications and systems are deployed in third party infrastructure providers. While the pros of this approach, including cost of operations and management, are many, the cons are rather dire. The main questions that arise are 1. Where is the processing unit located? 2. How many people are sharing this hardware? 3. How many have access to the data/code? The last question is quite complex. How does one trust the infrastructure provider? Even if one trusts the infrastructure provider and all of its employees with physical access or administrative access to the machines, one should still wonder about whether the other tenants using the same platforms are malicious or "honest but curious".

The solution is safe data processing and execution environments inside untrusted or com-

promised computers. They can be used for confidential data processing in public clouds where it is necessary to have protected, encrypted code and data to protect information from malicious administrators and hackers.

As software systems have such large code bases, the attack surfaces are proportionally large. Trusted Execution Environment (TEE) inverts this logic. A TEE as an isolated execution environment that provides security features such as isolated execution, integrity of applications executing with the TEE, along with confidentiality of their assets. TEEs makes it possible to run safe, smaller pieces of code on a trusted part of the processing unit. Such smaller code bases are called Trusted Code Base (TCB). As the attack surfaces are reduced, assuming there are no back doors, the confidence in the system is evidently higher. Since TEEs are mostly part of the existing hardware, this comes with its own security flaws. Just executing a TCB in isolation(enclaves) is not enough; one might need some kind of trusted hardware which is not part of the legacy software systems.

In the last quarter of 2015, Intel announced its release of commercially available machines with its proprietary Trusted Platform Modules (TPM), called the Intel Software Guard Extensions (SGX)[3]. SGX allows out of the box attestation mechanisms in conjunction with integrity checks, memory encryption and freshness guarantees. These checks help in situations where the privileged softwares like OSs and kernels are compromised. Given the extensive researches done on its implementations, SGX is still unmatched by its competitors.

This paper explores the TEE technologies, materialized with the design and evaluation of an Intel SGX based Machine Learning system, representative of the fundamental confidential computing. Focusing on the security and privacy of the model and underlying data, we wish to explore the following questions:

1. How can TPMs, in specific Intel SGX, help with data security?
2. What are the challenges in implementing a secure computational software system?
3. What benefits comes from confidential computing with SGX?

1.1 Definitions of key terminologies

This part explains the different terminologies [4] related to TEEs and its impacts on data security. This section is inspired by the background study performed in the field and also contains an abstract view of the different technologies. First and foremost, we have to distinguish TEEs, TPMs and similar technologies like *Hardware Security Modules(HSM)*.

We have already seen what TEEs are in the above section. Other classes of hardware for specialized cryptographic purposes exist along with TEEs, like TPMs and HSMs. TEEs are fundamentally different from TPMs and HSMs, although they can be implemented alongside the other hardwares.

A TPM is a specialized hardware chip designed to provide a "hardware root of trust" by using secret keys in such a way that physically trying to open the hardware or changing it to a different motherboard in order to retrieve its secret is next to impossible or, at the least, so that tampering is immediately evident. TPMs are not designed to provide general computational capacity. They do provide some basic computational capabilities like generation of random keys, encryption of small amounts of data, etc. This can also be done within a TEE, but it doesn't make sense as TEEs doesn't make a good hardware root of trust. On the other hand, TEEs are a better execution environment than TPMs as they are part of the primary processors.

An HSM[4] is an external physical device that specializes only in cryptographic operations, typically encrypting a plain-text with the key it holds and returning the cipher-text so the Operating System (OS) does not handle cryptographic keys. Like TPMs, they are designed to detect and/or make evident physical tampering, which makes them a useful tool to hold secrets in a safe place. They generally provide higher levels of protection than TEEs, but are separate modules to the main Central Processing Unit (CPU) and motherboard, accessed via bus, network, or similar.

Neural Network(s) (NN) are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. NNs interpret sensory data through a kind of machine perception, labeling or clustering raw input.

Confidential machine learning is a system where the confidentiality of the training and test data is secured. In this work, confidential machine learning also protects the confidentiality and integrity of the algorithm itself.

Confidentiality[1] preserves authorized restrictions on information access and disclosure, including means for protecting personal privacy and proprietary information.

Integrity guards against improper information modification or destruction and ensuring information non-repudiation and authenticity.

1. Data Integrity - The property that data has not been altered in an unauthorized manner. Data integrity covers data in storage, during processing, and while in transit.

2. System Integrity - The quality that a system has when it performs its intended function in an unimpaired manner, free from unauthorized manipulation of the system, whether intentional or accidental.

1.2 Structure of the Thesis

This report totally consists of 6 chapters. The first chapter is an introduction to this work, which also introduced key basic terminologies that are vital to understand the underlying technologies and principles.

The second chapter is the state of the art, which studies the latest and most relevant technological progresses in machine learning and existing confidential machine learning implementations and use cases. This chapter also explains the different types of TEE technologies in the market and gives a more vertical outlook on the top four TEEs. Finally this chapter concludes with the different attacks on TEEs and associated research works, which can be potential security considerations before indulging in this work.

The third chapter is a detailed background study on Intel SGX, including its functionalities, the crypto guarantees and primitives of this technology. The chapter concludes with Intel SGX enclaves, its associated lifecycle and Application Program Interface (API). This chapter solely emphasizes on Intel SGX.

The fourth chapter of this report consists of the methods and implementation of the practical work. The chapter has a brief introduction supported by use cases and the solutions that can be offered by this work. Next, an abstract view of the process is outlined, followed by the software design of the system and its individual modules. Finally, this chapter concludes with the most important part of this work, the enclave design.

[1] https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-12r1.pdf

The fifth chapter is a security analysis of the system as a whole. This chapter introduces the threat model and the attack surfaces of the system. This chapter also explores the authentication system used, the Windows SSPI and credential manager, and an analysis of its individual security features. Finally, the chapter includes the potential exploits that can be caused by side channel attacks to the system.

The final chapter of this report offers a conclusion and recommends possible future steps that can be taken for to achieve a more robust and flawless system for confidential machine learning.

Chapter 2

2 State of the Art

2.1 Introduction

In many application domains, multiple parties would benefit from combining their respective private data sets, to train machine learning models on the aggregate data and to share the benefits of the model usage. A good use case would be that multiple hospitals can pool in patient data to train a model to aid in diagnosing a disease. Machine learning stands on the principle that having more data allows the algorithm to produce a better model and performance. But building such a strong trust-based system requires a good implementation of information security tools like encryption, TEEs, etc.

There are other ways to implement this, like Fully Homomorphic Encryption (FHE) [5] a powerful cryptographic tool that can be used for privacy preserving machine learning. But many studies in the field report large runtime overloads, which limit their implementations in real world use cases.

First, this chapter will provide some background information on basic machine learning concepts followed by *confidential and oblivious multiparty computing* concepts and TEE technologies(Intel SGX). The chapter will conclude with an overview of popular attacks on TEEs.

2.2 Background Information

2.2.1 Machine Learning and Neural Networks

In the computer science world, machine learning is defined as the ability of systems to learn from data. Systems can be trained on data to make decisions. Training is a continuous process, where the system updates its learning and improves its decision-making ability with more data[10]. A novel example of a machine learning system is implemented by Hollywood producer Vinnny Bruzzese, who uses machine learning to predict movie revenues. It is to be noted that this work only focuses on neural network based machine learning and not other classical methods like KNN or K-Means, which are avoided to stay clear of redundant work.

Systems train(learn) in two ways: *Supervised* and *Unsupervised* Learning. In supervised learning, the system is given a historical data sample of inputs and known outputs, and it "learns" the relationship between the two using machine learning techniques(of which there are several). An example would be *Linear Discriminant Analysis (LDA)* where the system is given a set of labelled data(like vector representations of digits along with the label). On the other hand unsupervised learning includes reorganizing and enhancing the input data to find patterns in it and to place a structure on the unlabelled data. Cluster Analysis is a trivial example of unsupervised learning. Here the system tries to find a pattern among the individual data points and tries to cluster them with respect to their similarities. If viewed from the perspective of supervised training, these clusters can be the labels of the data points.

The most wide spread application of machine learning is in data science and data science focuses on predicting(one outcome) or forecasting(multiple outcomes) the future trends.

There are several machine learning techniques, as mentioned above. Most of these techniques were first introduced almost a century ago, as they couple strongly with mathematics. In the case of neural networks, the first idea of artificial neurons was presented in 1943 by neurophysiologist Warren McCulloch and mathematician Walter Pitts. In the case of K-Nearest Neighbours(KNNs), it ages back to the early 1960s. Current computing power makes it easier to implement these systems with ease and efficiency.

A *neural network* is an interconnected assembly of simple processing elements, units or nodes, whose functionality is loosely based on the animal neuron. The processing ability of the network is stored in the inter unit connection strengths, or weights, obtained by a process of adaptation to, or learning from, a set of training patterns[11].

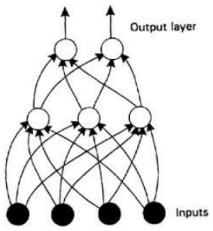


Figure 1: Simple example of neural network [11]

"Network" is used to refer to any system of artificial neurons. This may range from one single node to a large group of interconnected nodes. Figure 1 shows an example of a simple neural network. The nodes are arranged in a layered structure in which the input passes through two nodes before reaching an output. In the nodes, data is transformed using some rudimentary mathematical function. This is called a feed-forward neural network and is only one of several available. The synapses are modelled by a single number or a weight so that each input is multiplied by a weight. This product is then sent to the next layer of neurons. This is the basis of neural network based learning.

Let us consider an example of a simple node, with one input and one output. We use a simple sigmoid activation function to illustrate weights and how it affects the output.

Figure 2: A Simple Node

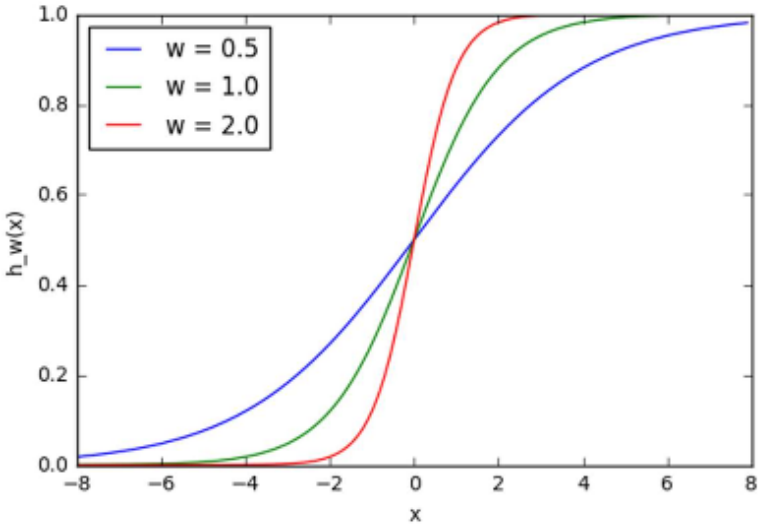


Figure 3: Effect of adjusting weights.

We can observe from Figure 3 that changing the weights, changes the slope of the sigmoid activation function. This is particularly useful if there is a need to model different strengths of

relationships among the input and output variables.

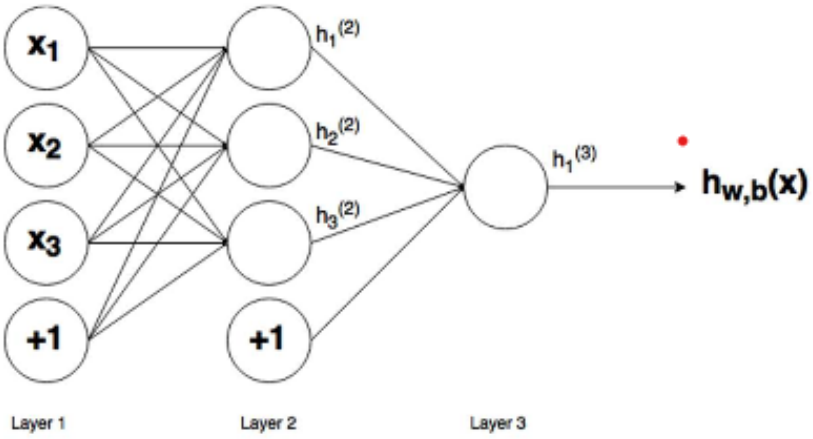


Figure 4: Three layer neural network

Neural network structures come in a myriad of different forms. The most common simple neural network structure consists of an input layer, a hidden layer and an output layer(Figure 4).

Feed-forward Pass

How can one calculate the output from the input values? The above 3 layer neural network, can be used to explain the math behind it.

$$h_1^{(2)} = f(w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + w_{13}^{(1)}x_3) + b_1^{(1)}$$

$$h_2^{(2)} = f(w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{23}^{(1)}x_3) + b_2^{(1)}$$

$$h_3^{(2)} = f(w_{31}^{(1)}x_1 + w_{32}^{(1)}x_2 + w_{33}^{(1)}x_3) + b_3^{(1)}$$

$$h_{W,b}(x) = h_1^3 = f(w_{11}^{(2)}h_1^{(2)} + w_{12}^{(2)}h_2^{(2)} + w_{13}^{(2)}h_3^{(2)} + b_1^{(2)}) \tag{1}$$

f(.) is the node activation function(sigmoid function here). $h_1^{(2)}$ is the output of the first node in the second layer and its inputs are $w_{11}^{(1)}x_1, w_{12}^{(1)}x_2$ and $w_{13}^{(1)}x_3$. They are simply added and then passed on through the activation function to calculate the output of the first node. The other 2 nodes follow similar calculation as above.

The final output of the node in the last layer is calculated from the weights(h_1, h_2, h_3) of the preceding layer and the weighted bias and not from the inputs (x_1, x_2, x_3). Therefore, we can observe in equation form the hierarchical nature of *Artificial Neural Networks (ANN).*

How can a neural network be used to classify an image of handwritten letters using this technique? The input consists of an encoding of binary data(dark and light values) of an image of one alphabet. To identify the letter class of the input, the output layer has 26 nodes, one for each letter of the alphabet. So when an input image is passed into the network, one neuron is fired whenever a pattern of the corresponding class is supplied at the input.

Back-Propagation

With supervised learning, we have training data which are labelled. So while training the network, we can compare the output of the neural network to our expected training value, y(z) and feasibly look at how changing the weights of the output layer would change the cost function for the sample (i.e. calculating the gradient). But a question arises on how to do this for all the hidden layers in between.

The method to answer this question is called back-propagation, which shares the cost function or "error" to all the weights in the network. In other words, it determines how much of the error is caused by any given weight.[12]. To calculate how much the weight has changed, say $w_{12}^{(2)}$ has on the cost function J, we use chain function illustrated as

$$\frac{\delta J}{\delta w_{12}^{(2)}}) = \frac{\delta J}{\delta h_1^{(3)}} \frac{\delta h_1^{(3)}}{\delta z_1^{(2)}} \frac{\delta z_1^{(2)}}{\delta w_{12}^{(2)}}$$

where $h_1^{(3)} = f(z_1^{(2)})$ (sub. in eq. 1) Further solving these partial derivatives by substituting appropriate values and further applying the chain rule of derivatives, we can calculate how much of a change in weight, say $w_{12}^{(2)}$ has on the cost function J for weights connecting to the outer layer. This can be generalized this as

$$\delta_i^{n_i} = -(y_i - h_i^{(n_i)}).f'(z_i^{(n_i)})$$

and the complete cost function derivative is :

$$\frac{\delta}{\delta W_{ij}^{(l)}} j(W, b, x, y) = h_j^{(l)} \delta^{l+1} \tag{2}$$

The outputs of the hidden nodes have no such direct reference as compared to the output nodes and they are connected to the cost function only through mediating weights and potentially other layers of nodes. So the term that needs to be propagated back through is $\delta_i^{(n_l)}$. This is the network's best connection to the cost function. So how much does a node in the hidden layer contribute to $\delta_i^{(n_l)}$ and how? The way a node contributes is through its weight $w_{ij}^{(2)}$.

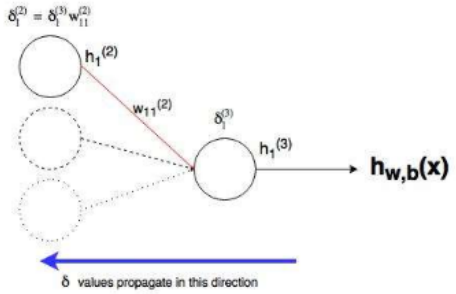


Figure 5: Simple Back-Propagation

As can be observed from Figure 5, the output layer δ is communicated to the hidden node by the weight of the connection. In the case where there is only one output layer node, the generalised hidden layer δ is defined as:

$$\delta_j^{(l)} = \delta_j^{(l+1)} w_{1j}^{(l)} f'(z_j)^{(l)}$$

In the case of multiple output nodes, the weighted sum of all communicated errors are considered to calculate $\delta_j^{(l)}$. Hence the generalised expression for the δ values for nodes in the hidden layers is given by

$$\delta_j^{(l)} = (\sum_{i=1}^{S(l+1)} w_{ij}^{(l)} \delta_i^{(l+1)}) f'(z_j^{(l)})$$

where j is the node number in layer l and i is the node number in layer l+1. The value S(l+1) is the number of nodes in layer (l+1). And then we calculate the complete cost function [Eq. 2].

Convolutional Neural Network (CNN) is one of the most popular deep learning methods with artificial neural networks. CNNs are widely used in the field of pattern recognition, from image processing to voice recognition. CNNs are so popular because of the lower number of parameters

compared to ANN. Hence CNNs were widely adopted to solve complex tasks which were not possible with classic ANNs. Another important aspect of CNNs is to obtain abstract features when input propagates toward the deeper layers. For example, in image classification, the edge might be detected in the first layers, and then the simpler shapes in the second layers, and then the higher-level features such as faces in the subsequent layers[13].

Convolution is defined as the process of moving the filter around the input image focusing on smaller parts called receptive fields. The first layer is the convolutional layer. For example, the top left corner of the image(say 5 * 5 * 3 filter), as the filter is convolving around the input image, it is multiplying the values in the filter with the original pixel values of the image. These products are summed up into the next layer. From an abstract point of view, these filters are thought of as feature identifiers. A simple example of a feature is a curve detector. If there is a curve in the image, the sum of the products mentioned above will have a larger value than of an image without a curve. The output of this convolutional layer is an activation map. The activation map will show the areas in which there at most likely to be curves in the picture.

There are many other layers that are interspersed between the convolutional layers. A classic CNN architecture would have the structure:

$$Input -> CL -> ReLU -> CL -> ReLU -> Pool -> ReLU -> CL -> ReLU -> Pool -> FC$$

where CL is the Convolutional Layer and FC is the Fully Connected Layer.

For the second convolutional layer, the input is the activation maps that result from the first layer. So each layer of the input is describing the locations in the original image for where certain low-level features appear. As the network is traversed and more convolutional layers are explored, we get activation maps that represent more and more complex features. By the end of the network, we may have some filters that activate when there is handwriting in the image, filters that activate when they see pink objects, etc. The Rectified Linear Unit (ReLU) layer is used to adjust or cut-off the generated output. This layer is applied to saturate the output or to limit the generated output. The ReLU creates a sparser representation.

$$ReLU(x) = max(0, x)$$

The pooling layer is used to downsample or, in other words, to reduce the complexity of further layers. The pooling layer has a similar effect to reducing the resolution. It does not affect the number of filters. A good example of pooling is *max pooling* where the image is partitioned to sub-regions and only the maximum value sub-region is returned to the next layer. Therefore, it should be applied only when the presence of information is important (rather than spatial information)[13].

The final layer is an important one. At this point, higher-level features are detected. The final layer takes the input volume and outputs an N-dimensional vector where N is the number of classes the algorithm has to choose from. For the same example of digit classification, the N is 10. So each value in the N-dimensional vector is a probability of a certain class. A fully connected layer looks at which high level features most strongly correlate to a particular class and has particular weights so that the computation of the products between the weights and the previous layer will get the correct probabilities for the different classes. CNNs mainly use back-propagation to adjust its

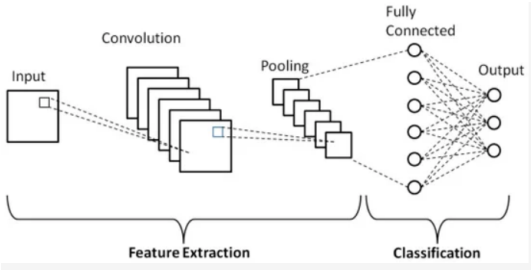


Figure 6: Convolutional Neural Networks

filter values. Hence a CNN model is trained. As for the testing part, a set of images with labels are passed into the CNN and the outputs are compared with the labels.

Another famous ANN architecture is the *Recurrent Neural Network (RNN)*. This type of ANN is widely used for learning from sequential data. The intuition behind RNNs is that the hidden state value is calculated as a combination of the previous hidden state and the latest input. Of course, each of these inputs is weighted depending on the algorithm. By always remembering the previous hidden state, RNNs are able to chain a sequence of events together. This also allows them to back-propagate errors to earlier time-steps during training. Bidirectionality can be incorporated into RNNs. Since RNNs are based on time-steps, if there was an error in any of the steps, the error signal has to be propagated back several time-steps. So through each of this step, the error signal is multiplied by its layer weight and might result in a drastic effect on the magnitude of the error

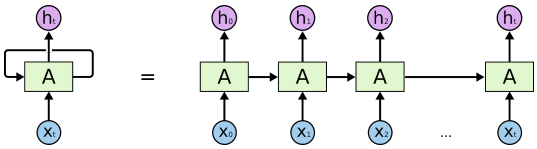


Figure 7: Recurrent Neural Networks

signal. This problem is called the *vanishing gradient problem* which makes it very hard to learn using a vanilla recurrent neural network. The same problem can occur when the weight is greater than one, introducing an exploding gradient, although this is slightly easier to manage thanks to a technique known as *gradient clipping*.

2.2.2 Confidential and Oblivious Machine Learning

Machine Learning consists of two stages, the training stage and the classification stage. Either one or both of these stages can be outsourced to the cloud, as illustrated by Thore et al[6]. In a traditional cloud-based machine learning, there are 3 different participants involved: the *Data Owner*, *the Cloud Provider*, and *the Content Providers*.

The data owner is the customer of the cloud service. He owns and/or is responsible for the data that has to be processed. Content providers are devices which upload data to the cloud and which have been authorized by the data owners. Remote devices and sensors are good examples of content providers. Using the same example of a hospital system, the patient, the hospital or some large company of lab technicians are the data owners. The cloud service may be run by a third party, a partner company, a research facility or even the company itself, off-premises or in some stand-alone facility. Content providers can be the health monitoring devices like scanners which directly send data to the cloud where it is processed.

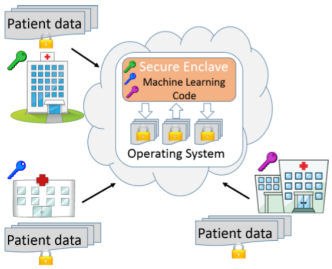


Figure 8: Use case: Hospital scenario

This kind of design is an ideal scenario for a trust based multiparty computation system. Thore et al's rationale for proposing these protocols relies on some scenarios where outsourcing computation to a third party cloud service makes sense from a practical and rational economic point of view. When the data is collected from multiple and diverse sources, an online service can host, store and compute this data without any interaction with the data owners. These services allow the data owner to access and query their data at any time using a device of minimal computational or storage capacity. The data owner can assign different privileges to other 3rd parties interested, to access the data or receive updates concerning some other processed form of the data.

Figure 8 illustrates a model of a use case in which a medical entity, for example a pharmaceutical company, can predict the occurrence of a disease. The medical entity collects data from clinical trials, but is simultaneously interested in outside data collected from different clinics around the country and internationally. The outside clinics, meanwhile, are very cautious with their patient data and therefore are not willing to send the data outside their jurisdiction. Yet these clinics do not have the technology available locally to train the model. Through a secure confidential and oblivious platform for machine learning, the pharmaceutical company is able to easily convince the clinics to participate in the research since no raw patient data ever leaves the premises. The pharmaceutical company is also able to securely link and train their model on the combined raw data without ever seeing it.

Another viable use case is in the insurance business. In this use case an insurer who is offering business interruption and property policies needs to better comprehend the current state of a supply-chain. So if the insurer is able to focus, for instance, on a port(air, land or sea) for goods, a holistic view of the location, content and value of all relevant goods enables the insurer to better understand the risks This also helps the insurer to offer current value-in-port policies and estimate damages faster, in case of a disaster and therefore optimize the release of reserves and improve consumer interactions. In this case, the required data are within the supply-chain participants. These participants are private corporations who compete against one another and are very reluctant to release their data to the insurer as these data may negatively impact their business, if the competitors got hold of them. A proper oblivious data sharing platform enables the insurer to give the supply-chain participants data privacy guarantees, thereby reducing the barriers to share data. Even though there is no computing needed here, this use case is of importance when it comes

to data transfer to a 3rd party.

Adding to the insurance use case above, an oblivious machine learning system can be used in Multi-company fraud detection without compromising confidentiality. Churn prediction and fraud detection are standard use cases in the banking and insurance industry. As explained above in section 4.5.2, any machine learning model performance correlates strongly with the available data. In particular smaller banks often suffer from too few data points to build robust and accurate models. Currently, with this paucity of data points and lack of data science expertise and technology, the need for a trust-based collaboration between such organizations is highly needed. The major catch is that banking and financial data are highly sensitive and cannot be simply shared between organizations. Hence, a confidential computing platform allows such organizations to collaboratively train machine learning models on sensitive data sets without any of the organizations having to trust each other with their data. So the more organizations participate in the system, the massively increased amount of training data can improve the fraud detection and churn prediction models. All participating organizations can use this result to optimize their existing system and hence reduce operating costs and mitigate risks.

When outsourcing computation to a third party service it makes sense to preserve data confidentiality by encrypting the data before uploading it to the cloud. This may limit the usage of the data. There are many recent advances in cryptography which allow searching on the encrypted data and perform simple operations without decrypting it. A good example of one of these scheme which allows arbitrary operations on encrypted data is the FHE scheme. FHE was first constructed by Gentry[5] and it was the basis for many more subsequent schemes which have become more practical with improved performances. Since encryption involves the addition of small noise terms into cipher-texts(padding etc), operating homomorphically on cipher-texts causes the inherent noise terms to grow and correct decryption is only possible as long as these noise terms do not exceed a certain limit. The noise growth is much larger in homomorphic multiplications than in additions. This means that FHE schemes that are in use right now can only evaluate polynomial functions of the data up to a bounded degree before the inherent noise grows too large.

The work done by Thore et al. proposes a protocol for confidential machine learning using homomorphic encryption scheme. Since many useful computational services only require evaluation of low-degree polynomials, so they can be deployed on encrypted data using only an FHE scheme

or subsequent SHE or LHE schemes[[7], [8], [9]]. Experiments were conducted with Linear Means Classifier (LMC) and Fisher's Linear Discriminant (FLD). The results showed computation time of 6 seconds with just 100 test and training data and 30 features.

The alternative to homomorphic encryption schemes is to have the machine learning service run in a trusted environment(TEEs). The following section will discuss different TEE technologies focusing on the advancements of Intel SGX.

2.2.3 TEE Technologies and Trust models

Trusted Platform Modules (TPMs)

TPMs are independent and are tamper-resistance co-processors. How is the root of trust established in a TPM? The root of trust comes from asymmetric keys also known as Endorsement Keys(EKs) which are integrated and burnt physically within the module. The main functional capabilities of TPMs are also mentioned in 1.1. Some implementations also provide trusted time and monotonic counters[2]. Internal components[2] of a TPM is illustrated are 9.

Who can access TPMs? Access to TPMs is completely based on ownership (who sets the shared secret). When ownership is set for a TPM, a storage root key(SRK) is created and is used for sealing data etc. Only the owner is fully capable of using TPMs to its full capacity remotely, although it may be factory reset through the assertion of physical presence, either legally or illegally. Nevertheless, users other than the owner can perform operations allowed by the owner, like querying software measurements, storing keys and using crypto primitives.

Being a separate component, TPMs are not a part of other isolation mechanisms such as memory page tables and privileged instructions. Security is therefore highly dependant on: 1. How TPM chips are wired to main processors, 2. What are the privileges users must hold to get access to such chips and 3. Whether adversaries might have physical access[2]. Security attacks on TPMs are analyzed in section 2.2.5.

ARM TrustZone

ARM TrustZone[15] is a popular TEEs used in Android phones and other low energy devices. It implements a protected enclave called "Secure World" which is different from the other parts (normal

[2]The components depends on the manufacturer of the TPM.

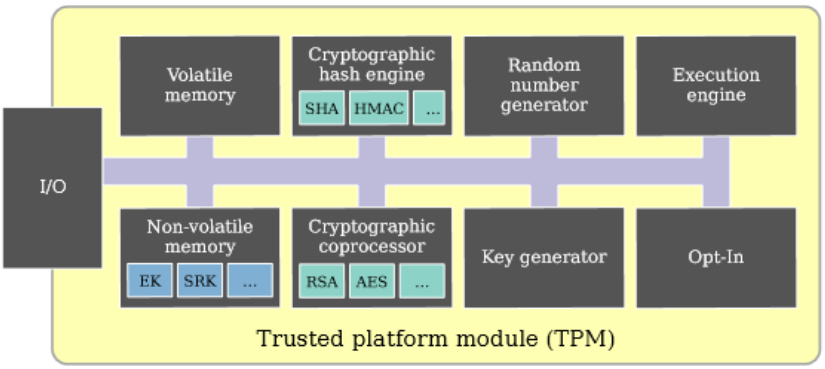


Figure 9: TPM components [2]

world). In this architecture, each of the processors or cores, is divided into two virtual ones: secure and non-secure. The context switch is made through a monitor mode (ARMv8-A), accessed by a special instruction, or through hardware exception mechanisms (ARMv8-M). During the boot, a secure firmware initialises the platform and decides what is part of the secure and non-secure worlds by configuring the interrupt controller and setting up memory partitions and peripherals. Then, the processor switches to the normal world, typically yielding control to the OS(Normal) bootloader[15], [16].

ARM has not specified any software attestation mechanism or data sealing directly but many implementations assume the availability of a secure hardware key. This is possible as ARM chip vendors are allowed to add their own IPs to the Socket on Chip(SoCs). So it is evident that a root of trust cannot be established solely with the ARM TrustZone but in conjunction with a TPM that can only communicate with the secure world.

Another disadvantage of the ARM TrustZone is that it provides only one secure world. So if there are multiple secure applications, they have to time share or divide the protected environment, which makes it more vulnerable to attackers who can use one partition. This can compromise the entire *secure world*. Alternatively, allowing only one secure application to run in the secure world is possible but is very restrictive. Moreover, the secure world is under the control of a separate OS, which means that it can still be accessed by some (possibly compromised) system administrator[16]

AMD Secure Encrypted Virtualisation (SEV)

AMD's SEV is designed to protect data in use by Virtual Machines (VM). It uses hardware encryption engine embedded in the Memory Controller (MC) which performs cryptographic operations. This method provides minimum performance impact and requires no changes to the application design. Key generation and management are performed in the AMD secure processor, a dedicated security subsystem based on an ARM Cortex-A5 processor physically isolated from the rest of the SoC. It contains a dedicated random-access memory (RAM), non-volatile storage in a serial peripheral interface (SPI) flash and cryptographic engines.

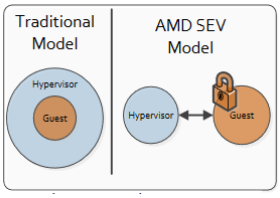


Figure 10: SEV Security Model [19]

Along with the SEV, AMD provides another TEE feature called the *Secure Memory Encryption(SME)*, which provides system-wide memory encryption using a single key. In contrast to SEV, which uses multiple keys (one per VM), when using SME, ephemeral keys are randomly generated at boot time by the secure processor and are used to encrypt memory pages that are marked by system software within page tables through the C-bit[17].

SEV involves the hypervisor in many ways. The main use of the hypervisor in SEV is that it manages the keys but has to access to them. This is done by the using address space identifiers(ASIDs) and manipulating them. ASIDs are initialized and specified when the VM is launched and are used to determine the encryption key. How are ASIDs used with key management? ASIDs are used as indexes into a key table in the MC. The the key table and the MC are responsible for key generation and storage. ASIDs are also used in cache pages to tag data for faster page hits.

AMD also introduced an updated version of SEV called AMD *Secure Encrypted Virtualization-Encrypted State (SEV-ES)*[18]. It encrypts all CPU register contents when a VM stops running. This prevents the leakage of information in CPU registers to components like the hypervisor, and can even detect malicious modifications to a CPU register state. Before SEV-ES, the processor register state of VMs was potentially exploited by hypervisors running under AMD SEV. SEV-ES

allows guests to restrict access to the register state, albeit possibly sharing it with hypervisors.

Although AMD SME/SEV/SEV-ES allow memory and registers encryption and platform attestation to a finer granularity (VMs), these technologies are vulnerable to memory integrity manipulation (e.g., rowhammer[20]) and rollback attacks, i.e., when a previous state (even if encrypted and signed) is replayed back and considered genuine.

2.2.4 Implementations with Intel SGX

Intel SGX [3] is a TEE by Intel which employs security guarantees with respect to memory freshness and integrity. SGX limits the secure execution to user level processes, which in turn allows to purge the OS out of the TCB. It also guards against memory tampering. This is done by a physical part of the processor which helps in bookkeeping. Because of this, the memory enclaves (protected memory) are considerably smaller in size(128 MB[21]). If there is a need to interact with the OS, there is also a considerable overhead involved.

SGX aims to guard the code execution against tampering and inspection(by infected OS) and also some physical attacks. This is done by some extension to the instruction sets in both x64 and x86 CPUs. Another feature of SGX is *Data Sealing* (Section 1.1). Data sealing is done by combining the data with integrity tags of memory in use by enclaves. This provides *authenticated encryption* and confirms the integrity of the plain-text. As a consequence, computations done in the enclave cannot be seen from the outside and any modification attempts are detected, including replay of previously authenticated values.

SGX also allows *remote attestation* 1.1 by which the enclaves are provided with certifications and keys. With remote attestation, a shared secret that enables a secure communication channel is established and permits the remote entity to provide the enclave with encrypted secrets.

There are many different implementations with Intel SGX. *Container-based virtualization* has been used to isolate applications for different platforms for interoperability. Many multi-tenant environments use Linux containers for performance isolation of applications, Docker [22] for the packaging of the containers, and Docker Swarm or Kubernetes [23] for their deployment. In multi-tenant environments, Linux containers like Docker and Kubernetes have a lower resource footprint, faster startup times, and higher I/O performance compared to virtual machines (VMs) on hypervisors. SCONE[24] is a secure Linux container based mechanism uses Intel SGX to prevent the container

from being vulnerable to outside attack. The work done by Sergei et al[24]. shows how SGX can be used to plug in the security vulnerabilities of traditional container-based implementations. SCONE boasts (1) a smaller TCB and (2) a low performance overhead. SCONE offers a secure C standard library interface that transparently encrypts/decrypts I/O data; to reduce the performance impact of thread synchronization and system calls within SGX enclaves. SCONE also supports user-level threading and asynchronous system calls.

Roland et al.[26], propose a new solution, *TensorScone*, to extend the functionalities of SCONE to support the widely-used machine learning framework, TensorFlow[25]. Their work assists in secure execution of existing applications that were built with TensorFlow without major changes in the commodity untrusted infrastructure. It also addresses the challenges in building such a complex system, overcoming the limitations of Intel SGX. The results show good performance with lower overheads while providing strong security with lower TCB.

Without proper confidentiality protection, input data to the cloud-based machine learning system might disclose user specific, sensitive information. The work done by Zhongshu et al.[27] focuses on the life cycles of input data instances in deep learning pipelines and identifies vulnerabilities of information leaks. They introduce a concept of ternary model partitioning based on 4 security directives established beforehand. The pseudo-code algorithm[11] by Zhongshu et al. showcases a basic working of confidential machine learning with Intel SGX.

2.2.5 Attacks on TEEs

TEEs are vulnerable to software flaws like memory safety violations, like *stack overflows* or *dangling pointers*. Such software vulnerabilities expose the data or can potentially change the enclave behavior. Attackers may find gadgets consisting in benign pieces of code (like the libc [28]) and reuse them to perform return-oriented programming (ROP[29]). These attacks prove that smaller TCBs have smaller attack surfaces. But using tools like SCONE[24], Haven[30], Graphene-SGX[31] propose complete run times in favor of usability as they need no modification of the legacy code. Software-based fault attacks shift the threat model from an attacker with physical access to the target device to a potentially remote attacker with only local code execution. Initially, these attacks were interesting in scenarios where the attacker is unprivileged or even sand-boxed. However, with secure execution technologies, such as Intel SGX, ARM TrustZone and AMD SEV, privileged

```
Input:  fn_enc      ▷ Encrypted FrontNet sub-model
        bn          ▷ BackNet sub-model
        lbl_enc     ▷ Encrypted model labels
        img_enc     ▷ Encrypted image input
        clt         ▷ Client key provisioning server address
        k           ▷ Number of returned predictions
############## Within SGX Enclave ##############
function SGX_LOAD_ENC_MDL_LBL (fn_enc, lbl_enc, clt)
    tls ← SGX_ATTESTATION (quote, clt)
    model_key, img_key ← SGX_GET_KEYS (clt, tls)
    fn ← SGX_VERIFY_DEC (fn_enc, model_key)
    lbl ← SGX_VERIFY_DEC (lbl_enc, model_key)
    frontnet ← SGX_LOAD_DNN (fn)
function SGX_INF_ENC_IMG (img_enc)
    img ← SGX_VERIFY_DEC (img_enc, img_key)
    ir ← SGX_MODEL_INF (frontnet, img)
    return ir
function SGX_CLASS_MAPPING (pv_k)
    result ← SGX_MAPPING (lbl, pv_k)
    enc_result ← SGX_ENC (img_key, result)
    return enc_result

############## Out of SGX Enclave ##############
function INITIALIZE_ENCLAVE (fn_enc, bn, lbl_enc, clt)
    eid ← INIT_SGX ()
    SGX_LOAD_ENC_MDL_LBL (eid, fn_enc, lbl_enc, clt)
    backnet ← LOAD_DNN (bn)
    return eid, backnet
function INF_ENC_IMG (eid, img_enc, k, clt)
    ir ← SGX_INF_ENC_IMG (eid, img_enc)
    pv ← MODEL_INF (backnet, ir)
    enc_result ← SGX_CLASS_MAPPING (eid, TOP (pv, k))
    return enc_result
```

Figure 11: Partitioned Deep Learning Inference Algorithm.

attackers must also be considered, as they are part of the corresponding threat models.

Since most of these attacks are focused on Intel SGX and its containers, this section covers different attacks on SGX. While some of these attacks are rectified by Intel, some still remain to be fixed.

Focusing on software attacks, the work done by Jo Van Bulck et al.[35] presents a *micro-architectural attack* that dismantles the security objectives of SGX by exploiting a speculative execution bug in latest Intel x86 processors. They successfully leaked plain-text enclave secrets from the CPU cache. As an optimization technique, processors may not choose to execute instructions (micro-operations) sequentially. Instead, they are executed out-of-order as soon as the required execution unit and/or any source operands become available. But towards the end of the process, these micro-ops are flushed by the processor back to the in-order and discard uncommitted micro-ops using a Reorder Buffer(ROB). Transient execution attacks abuse subtle micro-architectural side-effects to breach the memory isolation barriers. Concurrently multiple researches on transient execution attacks revealed fundamental flaws in the way the latest CPUs implement out-of-order execution. A type of Spectre[36] attacks exploit the CPU's branch prediction machinery to trick a victim protection domain into speculatively executing instructions out of its intended execution path. By "poisoning" the shared branch predictor resource, an attacker can steer the victim program's execution into transient instruction sequences that dereference memory locations the victim is authorized to access but the attacker is not.

Another research on Spectre Attacks by Guoxing et al. called SgxPectre Attacks[37] shows how to steal secrets from enclaves. Since Spectre attacks generally exploit speculative execution to subvert the confidentiality and integrity of SGX enclaves, the branch prediction of the enclave code can be influenced by processes on outside of the enclave and subsequently the enclave code can be potentially altered to execute instructions that lead to observable cache-state changes. An advisory observing the cache state changes can learn secrets of the internal registers or the enclave memory and hence defeat the confidentiality of an enclave. Furthermore, this can be used to identify the authentication tags used in AE and hence defeat the integrity of the data. The results of Guoxing et al[37] successfully applied SgxPectre Attacks to steal seal keys and attestation keys from Intel signed quoting enclaves.

Another class of attacks, called *Side Channel Attacks*, use alternative sources of information

like power, sound, electro-magnetic or time analysis to figure out the behavior of the target and finally reveal secrets. Computer systems now use complex OSs, VMs and other processes which may potentially leave traces that can be exploited by malicious agents[32]. One of these shared resources is the cache hierarchy, notably the most targeted component by TPM side-channel attacks [33]. There are many speculations that SGX is vulnerable to cache side-channel attacks. In fact, Intel does not consider side channels as part of the SGX threat model and thus states that SGX does not provide any specific mechanisms to protect against side-channel attacks (Intel 2016[45]). However, they also explicitly state that SGX features still impair side-channel attacks. Intel recommends using SGX enclaves to protect password managers and cryptographic keys against side channels and advertises this as a feature of SGX. Indeed, SGX does not provide special protection against microarchitectural attacks. Its focus lies on new attack vectors arising from an untrusted operating system.

Plundervolt[34] focuses on exploiting the dynamic frequency and voltage scaling features in modern processors to corrupt the integrity of the enclave computations. Plundervolt carefully controls the processor's supply voltage during an enclave computation, inducing predictable faults within the processor package. Consequently, even Intel SGX's memory encryption/authentication technology cannot protect against Plundervolt. The researchers behind Plundervolt managed to recover keys or induce memory safety vulnerabilities into bug free enclave codes implemented in real world implementations. Plundervolt showcases the failure of Authenticated Encryption (AE) and confidentiality of data and why AE is not enough to safeguard the integrity of general purpose computations.

Cache attacks exploit the timing difference between the CPU cache and the main memory. They have been the most studied microarchitectural attacks for the past 20 years. The works of Kocher [40]1996; Page[41] 2002; Bernstein[42] 2005; Percival[43] 2005 exposed vital flaws of CPUs and were able to derive cryptographic secrets. The work done by [38] was first of its kind to use cache-timing attacks to recover AES secret keys in less than 10 seconds when running inside an SGX enclave. This type of attack is called a *software-side channel attack*, where an advisory process thread is running on the the victim machine, trying to exploit any vulnerability. Schwarz et al[39] illustrates (Fig. 12) how one infected enclave exploits the confidentiality of the other tenants co-located on the same physical machine. This attack demonstrates a major vulnerability affecting SGX

by running a malware as an enclave code using the SGX features to conceal itself. The malware exploits micro-architecture implementation of CPUs, similar to Spectre[36] attacks, hence allowing the malware to infer information of other processes running on the same physical system. It is to be noted that since enclaves do not share memory with other enclaves, the OS or other processes and direct *flush+reload* attacks on enclaves are not possible. Same goes for any DRAM based attack from any outside processes as the hardware prevents any access to the DRAM rows in the Enclave Page Cache (EPC). However, an enclave can mount a DRAM attack on other enclaves as all the enclaves are located inside the same physical EPC.

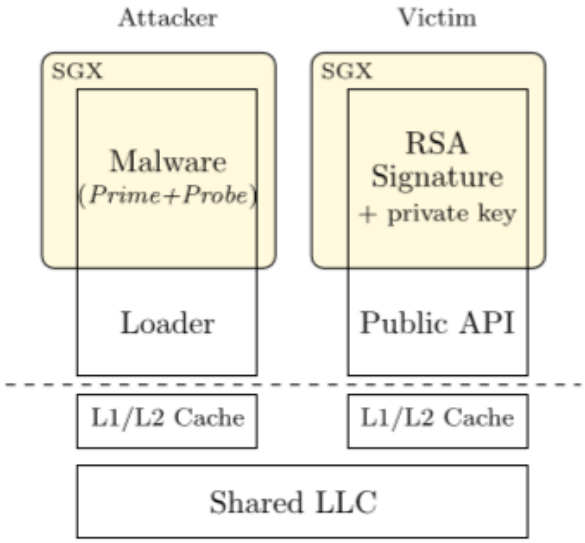


Figure 12: Threat Model of software based side-channel attacks - Cache Timing Attack

2.3 Conclusion

This chaper reviewed four important topics related to the thesis. The first part encompasses a brief introduction to machine learning, including the different types and techniques. Furthermore, the mathematics of two techniques of neural networks, namely feed-forward and back propagation, was explained and illustrated with examples.

The second part focused on oblivious and confidential computing and reviewed the current trends in confidential computing and its machine learning applications, tools and runtime environments. It is vital to know how to implement the different stages of machine learning efficiently,

whether to use cryptographic techniques (like HE, FHE, etc) or to use TEEs.

The third part is a study on the different TEE technologies in the market. This included TPMs and TEEs developed by leading chip makers like AMD, ARM and Intel. The different architectures and functionalities of these technologies were illustrated and studied along with their flaws. This also involved reviewing different enclave implementations, different run-times(container based) etc. Chapter 3 has the detailed working of Intel SGX as it is the technology selected for this work.

Different attacks on TEEs focusing on Intel's SGX were reviewed in the final section of this chapter. It provided a brief introduction to potential TEE vulnerabilities and attack models. This part described two main types of attacks: software based and side channel attacks. Both these attacks and leading research on them were analysed to get a view on the possible attack models and to get different perspectives of the vulnerabilities.

Hence this study shows a complete view of the current status and working of the different modules that will help in this work on confidential computing on machine learning and with Intel SGX.

3 Security Framework of Intel SGX

3.1 Introduction

Intel's Software Guard Extensions(SGX) is a set of extensions to the Intel processor architecture that aims to guarantee integrity and confidentiality to security sensitive processes by running them in secure containers or enclaves. There are certain information that are essential to start developing secure enclave based applications. Understanding what and how crypto guarantees are implemented with SGX is the first step towards developing secure enclave applications. It is also imperative to understand the lifecycle and memory management of enclaves to get a clearer view of how memory leaks or insecure pointer handling can result in the failure of crypto guarantees.

Section 3.2 and 3.3 of this chapter covers the security principles, crypto primitives fulfilled by SGX and other crypto modules built inside SGX. The next section covers the various functionalities of Intel SGX, like protected hardware and memory design, data sealing and attestation services. The chapter concludes with the SGX enclave life cycle.

3.2 Crypto Guarantees and Primitives

Cryptographic primitives are applied to software systems for security. These primitives have many assumptions and it is very difficult to build a secure system based on all these assumptions. Software attacks exploit bugs in a software components and also flaws in the architectural design of the software system. This section covers the cryptosystems used for this work.

The confidentiality of the message is protected when it is securely transmitted over an insecure medium without an adversary eavesdropping on it. The integrity of the message is guaranteed when the receiver obtains the message with a guarantee that it has not been tampered with or notice that some foul play was involved during transit.

Freshness guarantees assures the receiver that the latest message is received from the sender or will notice an attack. This is used in the case of multiple messages transmitted over an untrusted medium. A freshness guarantee is stronger than the equivalent integrity guarantee, because the latter does not protect against replay attacks where the attacker replaces a newer message with an older one coming from the same sender[45].

It is also important that while signing an enclave the author chooses to remain private.

Zero knowledge proofs aid in reserving this secret through a series of information sharing.

This section explains crypto guarantees and their associated primitives. Finally the respective crypto algorithms are listed and explained in detail.

Guarantee	**Primitive**
Confidentiality	Encryption
Integrity	MAC / Signatures
Freshness	Nonce + integrity

Table 1: Guarantees and Associated Primitives

3.2.1 Zero-Knowledge Proofs

Direct Anonymous Attestation (DAA) is a crypto primitive which enables remote authentication of a trusted platform whilst preserving the privacy of the platform user. Intel SGX used the EPID 2.0 implementation of DAA to attest for an enclave. DAA has been adopted by the Trusted Computing Group to address this same issue of preserving the identity of the enclave user.

The DAA protocol can be generalized based on three entities, namely the DAA member, the issuer and a verifier. The issuer has the responsibility to verify the trusted platform in the beginning and to issue (through the Join process) the DAA credential to the platform. The member (the platform itself) uses the DAA credential from before to sign. And finally the verifier confirms the credential without violating the platform's privacy. The verifier uses zero-knowledge proofs to perform this verification of the trusted platform.

Zero-knowledge proof provides a way for one entity to prove that it knows a secret without conveying anything else. Zero-knowledge proofs acknowledge that it is difficult to demonstrate that one possesses knowledge of certain *information* without simply revealing the *information* and allows the entity to prove possession without revealing the *information* itself.

How does Intel implement this with its EPID? We know that Intel SGX enclaves are authenticated using the EPID cryptosystem. EPID is a group signature scheme that is intended to preserve the anonymity of the signers. Now to put the *information* in context of who the three entities are:

1. DAA Issuer - Intel's Key provisioning service.

   Intel, as the issuer, is the only party that knows the issuing signing key, also called the *Intel*

Signing Key(ISK). The *Group Public Key (GPK)* is published within a group. Public key certificates signed by the ISK and is verifiable using the associated public key. This public key is different from the GPK.

2. DAA member - The enclave that has to be attested.

   The members join the group through an interactive protocol with the issuer. This is designed in such a way that the issuer is not aware of the members' private keys. Given an enclaves report, a member creates a quote that includes an EPID signature of the report. The quote acts as an attestation that the enclave is running on a trusted SGX platform.

3. DAA verifier - The remote service or the user (Figure 21). They verify that a given MRENCLAVE is running on a given trusted hardware.

The issuer publishes a group public key while securely storing the master issuing key. After an enclave authenticates itself to the provisioning service (issuer), it generates an EPID member private key. This private key is the attestation key and hence joins the group. This key is used to sign quotes.

Initially, a provisioning secret is stored in the e-fuses of each SGX-enabled processor[45]. This can potentially be used by Intel to trace individual chips during the enclave authentication phase. But the Join protocol can be blinded. Then the provisioning service cannot trace a signature to an attestation key or the chip.

3.2.2 Confidentiality

As mentioned above, confidentially is attained when an advisory can intercept a message but cannot learn any information from it. This is typically achieved by encryption, a crypto primitive. The sender can encrypt the information using various techniques and the receiver at the other end can inverse this function and read the message. The most common encryption methods are block ciphers, which are used in this work.

There are two types of encryption techniques: symmetric and asymmetric. Symmetric key encryption algorithms use the same secret key for both encryption and decryption. This is shown in Fig. 13.

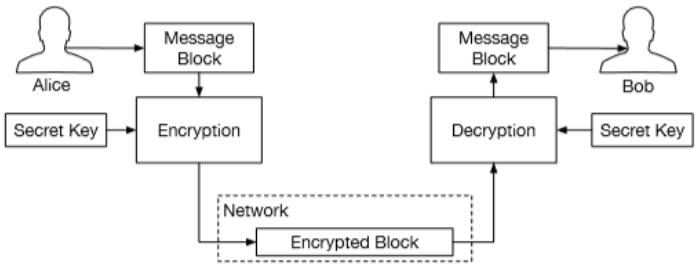


Figure 13: Symmetric Encryption

Asymmetric encryption uses 2 different keys, one for encryption and one for decryption. A public key is used for encryption and a corresponding private key is used for decryption. This is illustrated in Fig. 14.

The most popular and inherently secure block cipher based symmetric encryption is the Advanced Encryption Standard (AES). AES operates on 128-bit blocks using 125 and 256 bit keys. AES transforms any 128 bit block to another 128 bit block using secure permutations and transformations. The working of AES is out of the scope of this work. Block ciphers alone do not guarantee confidentiality by their own, so they must be combined with other operating modes. The most commonly used mode is the Cipher Block Chaining (CBC). This operating mode splits the message into 128-bit blocks, encrypts the blocks, and then attaches them to the subsequent block encryption.

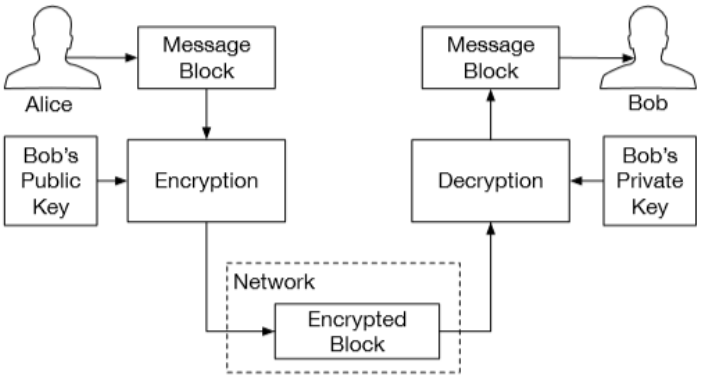


Figure 14: Asymmetric Encryption

Another example of operating mode is the counter mode(CTR). The counter mode turns a block cipher into a stream cipher. The next key-stream block is generated by encrypting successive

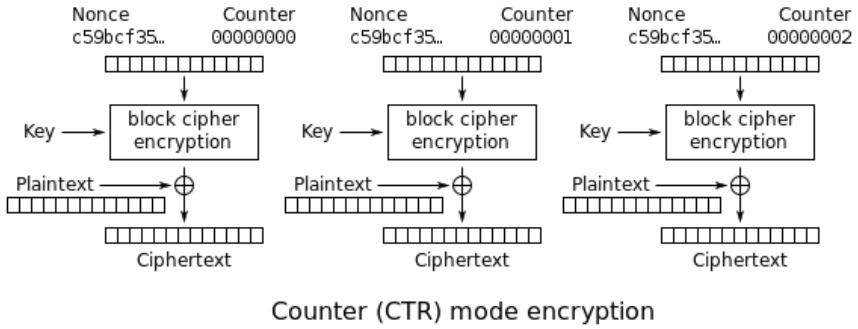


Figure 15: CTR mode Encryption

values of a counter. A counter can be any function which produces a sequence that does not repeat for a long time. The easiest example for a counter function is an increment operator. Unlike CBC, which is sequential, CTR is capable of encrypting blocks in parallel. CTR modes uses an Initialization Vector (IV) and a nonce for freshness. Freshness is discussed in detail in section 3.2.5. With CTR, the IV or nonce is combined with the counter to produce an actual unique counter block for encryption. Fig 15 and 16 [3] shows the encryption and decryption process of CTR block cipher mode of operation.

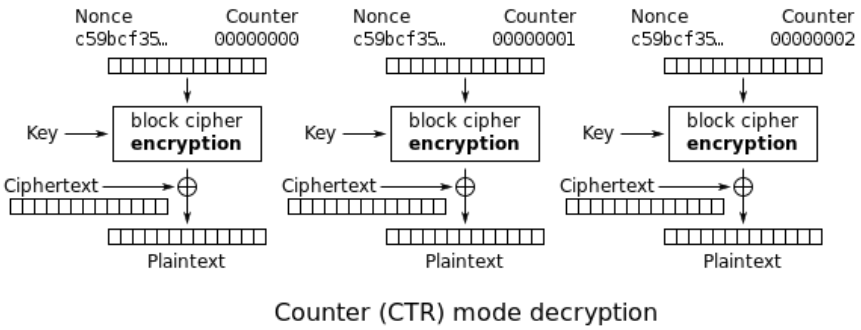


Figure 16: CTR mode Decryption

3.2.3 Key Management

All cryptographic primitives rely on keys. Security analysis mainly focuses on ensuring that these keys are generated, handled and deleted according to the primitives' assumptions[45]. The vital

[3] https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

part of any key generation algorithm is the usage of random data to produce a maximum entropy key. A shared secret if key is established and used by all the parties in symmetric key cryptography. Here one party generates a secure private key using the key generation algorithm, then transmits it securely to the other parties. The transport medium should be secure as such it should provide confidentiality and integrity guarantees. This is a large logistical burden. The algorithms used are specified in the sections below.

3.2.4 Integrity

Integrity is a guarantee that the data did not succumb to any changes by a malicious third party when it was transmitted through a not necessarily secure medium. Just because the data was encrypted does not guarantee that the integrity of the data was not compromised. At the centre of integrity guarantees is a family of functions called Secure Hashing Functions(SHA). These hash functions operate on an *unbounded* size of data and put out a small fixed-size output. Hash functions provide pre-image resistance, which guarantees that a malicious party cannot reverse engineer the input data from the output.

The most popular secure hash function is the Secure Hashing Algorithm(SHA)[47]. The most secure version of SHA is the 256-bit SHA-2 algorithm. This algorithm is used in this work. As mentioned above, SHA is a family of block hash functions that takes a fixed sized blocks as input. The internal state of SHA algorithms is fixed state too, meaning a fixed block size is operated upon at one time.

How can integrity be preserved with hashing functions? Similar to the encryption/decryption schemes (Section 3.2.2), integrity checks also function with either a symmetric or an asymmetric key setting. With the symmetric key setting, a Message Authentication Code (MAC) is calculated. The MAC of an input message is calculated with a symmetric key. The output of the MAC is fixed length and does not change for the same message and key. So at the receiving end which has the key, the MAC is recalculated and compared with the MAC that was received. The SHA algorithm is used to calculated the MAC of a message and such a cryptosystem is called a Hash Message Authentication Code(HMAC). The working of a HMAC is illustrated in Fig 17. Here integrity is assured by computing a HMAC and transmitting it over a network along with the message. The receiver recomputes the HMAC and compares it with the version received from the sender. So if an

adversary manipulated the message in transit, the receiver-calculated MAC and the MAC received will be different.

With an asymmetric key setting, integrity guarantees are provided by a crypto primitive known as signatures. Since it is an asymmetric key system, there are private and public keys. The sender provides a private key to a signing algorithm which generates a signature. The message and the signature are sent. At the receiver's end, a signature verification algorithm verifies the signature using the senders public key. Only small messages can be signed directly efficiently and cheaply. Hence the message to be transmitted is first passed through a hash function which need to be cryptographically strong, and the hash is provided as the input to the signing algorithm.

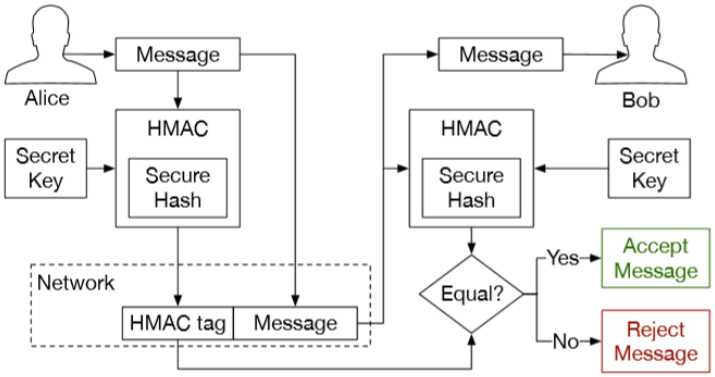


Figure 17: Hash Message Authentication Code(HMAC)

3.2.5 Freshness

Freshness *guarantees* are combined and built over an integrity *guaranteed* system. This is done by adding a unique piece of information to each message. Freshness is guaranteed by the use of *Nonces*. Nonces are single-use random numbers. Nonces solve the overhead of storing this random data on the sender side because it only needs to store them on the receiver side.

A nonce is usually combined with the message timestamp and expiration scheme. This expiration scheme discards the expired messages and hence saves the receiver a lot of memory space. This scheme depends on synchronization among all the entities participating in the communication. Alternatively, nonces can be used in challenge-response protocols for the same reason as above. The challenger generates a nonce and attaches it to the challenge request. The response to the challenge

request includes an acknowledgement of the embedded nonce, so the challenger can distinguish between a legitimate new response and a replay attack.

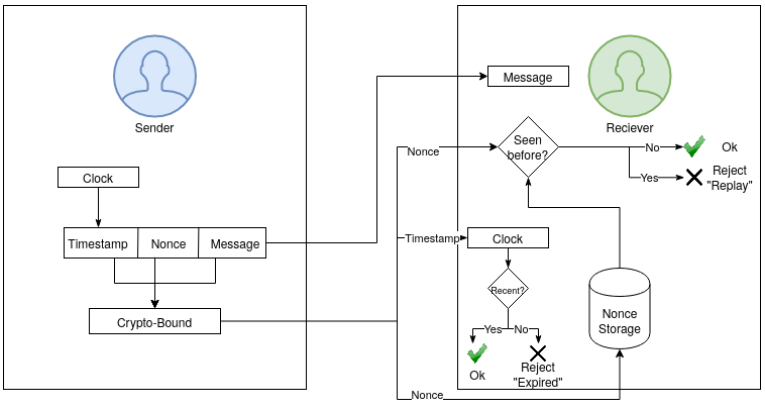


Figure 18: Freshness Guarantee using Nonces

3.2.6 Random Generator

SGX uses cryptographic primitives everywhere, be it to sign an enclave, to seal encrypted data off from the enclave, or to attest software. SGX also allows enclaves to use cryptography by providing a crypto libraries to enclave authors. The security of these primitives relies on a secure, random number that is used in these schemes. A good source of randomness (entropy) is necessary to build a high-quality random number generator. Commonly, C++ uses rand() function to generate random numbers. The rand() implementation is deterministic and therefore capable of generating predictable cryptographic secrets. This is why SGX does not support rand()[44].

SGX's implementation of a random generator method is the :

```
sgx_read_rand(unsigned char* rand, size_t length_in_bytes);
```

This function is called the hardware-based pseudo-random generation(PRNG), through the RDRAND instruction, which is available in Intel CPUs. The RDRAND instruction generates a random number by provisioning hardware implementation of the underlying Digital Random Number Generation(DRNG). It is possible to call RDRAND directly, but it might fail sometimes. So the SGX random generator function sgx_read_rand() calls RDRAND but has a retry limit of 10 times.

There is one caveat with SGX's PRNG function. As stated in the SGX developer manual[44],

RDRAND and RDSEED are non-SGX instructions that need to use the CPU. So malicious hypervisor can force a virtual machine exit preventing an enclave program from using the random function.

3.2.7 Crypto Algorithms

SGX uses the following cryptographic algorithms.

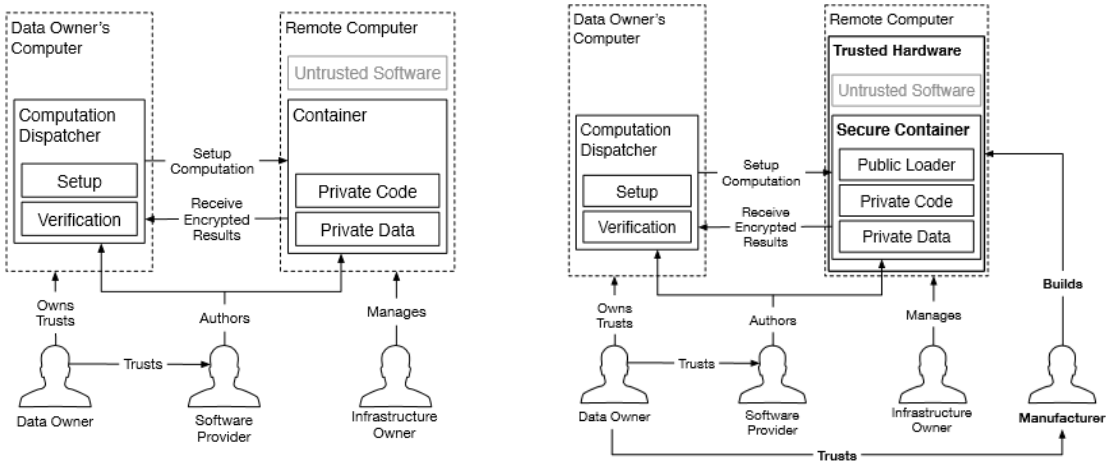
1. RSA-3072 PKCS#1 v1.5 with SHA-256, to compute enclave signatures.
2. RSA-2048 PKCS#1 v2.1(OAEP) with SHA-256, for attestation process.
3. ECDSA signatures over p256 NIST curve, with SHA 256 for enclave policy checks.
4. EC- Diffie-Hellman over p356 NIST curve for key exchange.
5. AES-GCM for data sealing
6. AES-CMAC for key derivation.
7. AES-CTR for memory encryption.

SGX only has support for AES-128 bit encryption. RSA schemes provide a lower security of the order 112-bit for RSA-3072, and of the order 96-bit security for RSA 2048. But these algorithms are standardized, trusted and time-tested. So if there is any security flaw, it will be in their implementation.

A good example is the cache-timing attack (Section 2.2.5) which makes the software implementations of AES very vulnerable. To counter this, some AES implementations in critical SGX components combine hardware instructions (Advanced Encryption Standard New Instructions, AES-NI) to perform a series of rounds and a table-based implementation, optimized to minimize data-leaks from this attack. In the Linux implementation of AES, a slower version of AES is used without AES-NI but with other kinds of cache-timing attack mitigation mechanisms.

3.3 SGX functionalities

In a traditional remote computing environment, the user relies on the cloud provider, its administrators and their hardware, to perform computations on their data. Using SSL/TLS and similar tools,



(a) Remote Computing

(b) Trusted Computing

Figure 19: Traditional vs Trusted Remote Computing - An abstract view

the user has some assurance of the computation's integrity and confidentiality, affecting data in transit. So if the cloud service provider uses a trusted hardware, it establishes a secure container, and the remote computations service user can move the data and computations to the secure container. The trusted hardware, using the below explained functionalities, protects the data's confidentiality and integrity while the data undergoes computation. The user can trust the hardware that is in the remote computer and entrust their data to a secure container which is hosted in the secure hardware.

An SGX-enabled processor can effectively isolate the enclave's code and data from the outside environment and privileged software and hardware devices attached to the system bus, thus protecting the integrity and confidentiality of the enclave and its data. This means that the code that runs inside an enclave cannot communicate directly with the user, i/o devices or SMBs. SGX model is still compatible with the traditional software by layering in the Intel architecture, where the OS kernel and hypervisor manage the computer's resources[45].

3.3.1 Memory design

SGX sets aside a region in the Random Access Memory(RAM), called the *Processor Reserved Memory(PRM)*. This region of the memory acts as the base of all enclave computations. The CPU protects this region from all non-enclave processes and memory access. As mentioned above, these

restricted processes include the kernels, hypervisors and, mainly, the OS.

The PRM holds the *Enclave Page Cache(EPC)*, which encompasses 4 KB pages that store both the code and data. Even though the EPC pages are assigned to the enclaves by system software, this is protected, as only the EPC Metadata is disclosed to the software once the enclaves are initialized. Each EPC page belongs to exactly one enclave. Once all the pages are loaded into the EPC, the system software asks the CPU to change the status of the enclave as *initialized*, after which the enclave is able to run the application code inside it. The loading method is disabled as soon as the enclave is *initialized*. The enclave contents are then hashed by the CPU. This is called the measurement hash, which is used by attestation services.

This aims to provide confidentiality for the code and data that run in a remote environment, as not even the hypervisor, which partitions the resources to the cloud users, can access a protected enclave. The PRM acts as a secure storage mechanism for the data.

3.3.2 Data Sealing

Data sealing is a process of enabling data that belongs to a TEE to be bound to it in such a way that it can only be restored if the same TEEs and TCBs are restored. When an enclave is initialized, it gives protection to the data by keeping it within the boundary of the enclave. Since the enclave memory is restricted, it is important to identify enclave secrets from others which need to be preserved through the following events.

1. The application is done with the enclave and closes it.
2. The application is closed.
3. The platform on which the application and the enclave run is hibernated or shutdown.

What happens to the data that is in the enclave buffers after the enclave is closed? The general process requires the secrets in the enclaves to be destroyed when the enclave is closed and hence they are lost. So if there is data that need to persist even after the enclave is shutdown, said data need to be stored outside the enclave before it is closed. Obviously the data need to be encrypted before being stored outside the enclave. But how does this encryption work and how are the keys generated and managed? SGX has a mechanism which retrieves a key unique to the

hardware platform and to the enclave, to encrypt the data. These processes of encrypting and decrypting enclave secrets are called sealing and unsealing, respectively.

Intel provides two different policies for sealing data. These are the conditions that need to be met when the data is to be unsealed.

Seal to the current enclave

As we have seen in the above section 3.3.1, measurement hashes are generated by enclaves by hashing the enclave memory. Sealing to the current enclave uses the current version of the enclave measurement taken when the enclave was created and binds the measurement hash to the encryption key used by the sealing operation. A specific instruction, EGETKEY, does this binding.

Subsequently, to decrypt these data only an enclave with the same measurement hash will be able to unseal this data. The logic is that if the sealed enclave file is tampered with, the enclave's measurement will change as well and the data cannot be restored.

Seal to the enclave author

As opposed to the above section on sealing enclave data with the enclave measurement, this policy of sealing uses the enclave author's signature (stored in the MRSIGNER register) and binds it to the key used to seal the data. This binding is done by the hardware EGETKEY instruction. This policy not only binds the enclave author's signature, but also the ID of the enclave. Enclave IDs are generated when the enclave is instantiated[44]. This method of sealing data with enclaves makes sure that only an enclave with the same value in the MRSIGNER register and the same ID will be able to unseal the data.

Which of the two sealing processes is more secure? There are two main benefits of sealing enclave data to the enclave author(2^{nd} policy). First, it allows for an enclave to be updated by the enclave author without serious upgrade process or recalculating measurement hash, as is with the first policy. Therefore it is quite a bit easier to reseal the data to a newer version. Secondly, it allows enclaves from the same enclave author to share data seamlessly.

Another subtle feature in "SGX seal to enclave author" includes setting a Security Version Number (SVN) to an enclave. The CPU stores this SVN number and uses it to generate a seal key for an enclave. So an enclave has to provide this number when requesting to seal/unseal data. This

makes sure that there is an option to unseal data that has been sealed by an earlier version of the enclave. This feature can mainly be used for software updates.

Sealing Process

An overview of the process of sealing data within an enclave is as follows.

1. The first step is to allocate memory in the enclave to store the encrypted data and the sealed data structure. The SGX sealed data structure has the payload of the data to encrypt and the Additional Authentication Data (AAD). AAD is the additional data that will not be encrypted but for which a MAC will be calculated. AAD can include additional information, like the software version, headers, and so on.

2. The second step is to call the necessary seal data API (Section 3.2.7). The data is sealed using AES-GCM.

3. Finally, the seal data structure (including the key request structure) is *persisted* to decrypt the data.

Unsealing Process

An overview of the process of unsealing data within an enclave is given below.

1. Allocate memory to hold the decrypted data.

2. Read the persisted key structure which was used to seal the data.

3. EGETKEY instruction is called with the key structure to obtain the seal key.

4. The unsealing algorithm API is called to decrypt the data using the above seal key [4] [5]

Two running instances of the same enclave can be distinguished at the time they attest. But there is no SGX mechanism to prevent one enclave instance to have access to the sealed data of another enclave, as they both call the same EGETKEY instruction. This mechanism enables data

[4] It is imperative that the seal key is deleted from memory to prevent accidental leaks

[5] If AE is used, the hash tag generated by the decryption algorithm is matched against the tag generated during the encryption.

to be kept secret across power cycles[44]. This can be potentially fatal because if one instance of the enclave is compromised, it compromises all the data sealed by all instances of the enclave.

A code snippet for sealing and unsealing data is as follows.

```
#include <iostream>
int main()
{
    sgx_status_t res;
    uint8_t* plaintext = (uint8_t*) malloc(plaintext_len);

    // Allocate space for sealing
    uint32_t ciph_size = sgx_calc_sealed_data_size(0, plaintext_len);
    uint8_t* sealed = (uint8_t*) malloc(ciph_size);
    uint32_t plain_size = plaintext_len;
    // Seal  the data
    res = sgx_seal_data(0,
                        NULL,
                        plaintext_len,
                        plaintext,
                        ciph_size,
                        (sgx_sealed_data_t *) sealed);
    assert(res == SGX_SUCCESS);
    // unseal the data
    res = sgx_unseal_data((sgx_sealed_data_t *) sealed,
                        NULL, NULL, plaintext, &plain_size);
    assert (res == SGX_SUCCESS);
}
```

3.3.3 Software Attestation

Software attestation is the process of proving that a software is running on a secure and authenticated platform. In the process of demonstrating that the software has been established on a secure platform, a 3^{rd} party establishes that the software is running inside an secure enclave on an Intel SGX-enabled platform prior to provisioning that software with secrets and confidential data. The attestation process relies on the ability of a platform to demonstrate its validity by producing a credential that accurately reflects the signature of an enclave. This signature also has information about the enclave's security properties.

Intel's developer guide[44] mentions two mechanisms of attestation.

- Mechanism 1: Local Attestation between enclaves running on the same hardware (intra-platform).
- Mechanism 2: Remote Attestation between an enclave and a remote 3^{rd} party (inter-platform).

Local Attestation

This mechanism for attestation is mainly used when different enclaves inside the same computer need to co-operate with one another to perform some higher-level function. Hence one enclave has to prove its identity and authenticity to the other. Initially, both enclaves create a credential, also known as a report. This report contains a cryptographic proof that the enclave is in the specified platform. A hardware-generated key is used to get this proof. This report can be exchanged among the enclaves to prove that they exist on the same system.

This authenticated encryption mechanism uses a symmetric key system, in which only the enclaves on the platform that creates the report know the key (hardware generated), which is only embedded in the hardware platform.

A report consists of the following data:

- Measurement of the code and data in the enclave.
- Hash of the public key in the ISV certificate (the enclave receives this at initialization time).
- User data (AAD).
- Any other security-related data like key blobs.

- A platform verifiable signature of the above data.

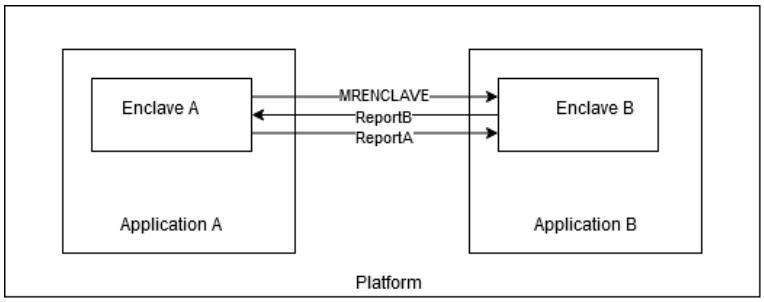


Figure 20: Intra-platform Enclave Attestation Mechanism

Figure 20 shows an example of how two enclaves can authenticate each other on the same platform.

1. Application A hosts enclave A and application B hosts enclave B. Both the applications are on the same platform. Enclave A sends its MRENCLAVE ID to enclave B.

2. Enclave B asks the hardware to produce a report destined for enclave A using the MRENCLAVE value it received from A. B transmits this report back to A.

3. Enclave A receives the report from B and requests the hardware to verify the report using its MRENCLAVE. If this is successful, it proves that enclave B is on the same platform as A. Enclave A then reciprocates this by generating a report using enclave B's MRENCLAVE value which was inside the report it received and transmits it to B.

4. Enclave B then verifies the report to confirm that enclave B exists on the same platform as A.

It is important to note that all software, including the applications outside the enclaves, is untrusted. So local attestations can be useful for verify enclaves within a single application as well.

Remote Attestation

Remote attestation confirms that the user is communicating with a secure enclave hosted on a trusted hardware. An attestation signature called a quote, is produced by the hardware's secret attestation key and is presented as *proof* to the user. The *quote* covers the container's initial state, a challenge nonce produced by the remote computer, and a message produced by the container.

An application that hosts an enclave can also ask the enclave to produce a report, which will then be used by the remote platform service to produce a quote which reflects the enclave ID and the platform state. This quote is verified using Intel(R) EPID signature verification. Hence, unlike local attestation, the CPU key is not directly exposed outside the platform.

A quote includes the following data.

- Measurement of the code and data in the enclave.
- Hash of the public key in the ISV certificate. The enclave receives this at initialization time.
- Product ID and the Security Version Number(SVN) of the enclave.
- Enclave attributes, like information concerning release or debug mode.
- User data. Similar to a local report, this establishes a secure channel in the attestation process so that the remote server can pass secrets to the enclave that has been attested.
- A signature of the above data which is signed by the Intel EPID key.

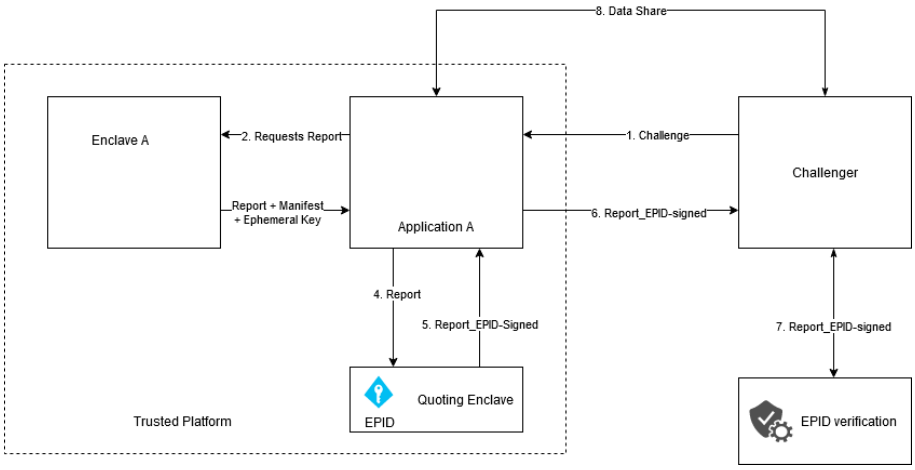


Figure 21: Inter-platform(Remote) Enclave Attestation Mechanism

The steps shown in figure 21 are as follows:

1. The challenger (cloud/service provider) issues a challenge to the application after it receives a communication request from the application which has the enclave. The challenge contains a nonce to make sure the communication is timed and live.

2. The application requests a report from the enclave inside it. The nonce received from the challenger is sent along with the request.

3. The enclave then generates a report structure similar to the one in Local attestation. This report also contains a manifest structure. A manifest can have additional information like the nonce, user data portion of the report and an ephemeral public key in case the challenger needs to send back some secrets.

4. This report is then signed (using the Intel EPID key) by the quoting enclave after it authenticates the report.

5. The application receives the quote from the quoting enclave.

6. The quote, along with the manifest is sent to the challenger.

7. The challenger uses the EPID public key certificate to validate the signature of the quote. This is done with the help of a EPID verification service provided by Intel.

8. Next the challenger compares the data in the quote against the trusted configuration. If it is a match, the challenger provides the necessary service to the application.

Note: The challenger can also add additional constrains, like version number of the enclave, measurement of the data/code, the product ID, Author ID, etc. This is made for cases which sound trivial but are a real concern. For example, an enclave developed in debug mode should never be trusted with any secrets.

Figure 22 puts the whole process in the perspective of how the user can communicate with a remote cloud service by confirming the identity of the remote enclave and share bidirectional encrypted data. The *Diffie-Hellmann Key Exchange* depicted in the figure, is used to establish a secure communication medium though which the hash of the initial state (enclave report of the enclave) is sent from the cloud to the data owner. Following the authentication of the enclave in the cloud using this hash value, the data owner can share the confidential code and/or data with the enclave in the cloud.

In this use case the data owner is the challenger and their enclave is established on a trusted cloud-based platform using the manifest explained above, where the application and the data owner exchange an ephemeral public key. After the trusted platform is attested, using the EPID of the

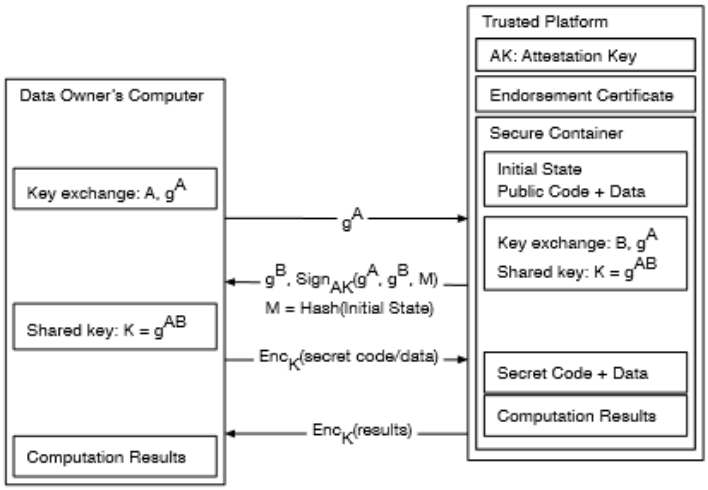


Figure 22: Inter-Platform communication using Remote Attestation Mechanism

enclave, both the user and the enclave can communicate through an encrypted channel using this key and any public key cryptography.

What incentive does the cloud provider have to build this secure platform? If there is a high demand by tech savvy customers to rent secure containers in the cloud provided by trusted hardware and authenticate the software by attestation mechanisms, the cloud providers will be forced to include this technology. This also forces them to build the containers according to specifications so that the containers pass the attestation.

3.4 SGX Enclave Life Cycle and APIs

The life cycle of an Intel SGX enclave is intertwined with resource management, specifically the allocation of the EPC pages. Hence, this section explains the major steps in an enclave's life cycle. Some of the major steps of an enclave life cycle are: Creation, Loading, Initialization and Tear Down. It is important to study these steps to effectively work with SGX enclave development.

3.4.1 Creation

There are specific instructions that handle and manage the enclave life cycle. The ECREATE instruction creates a new enclave by turning a free EPC page into a SGX Enclave Control Struc-

ture(SECS). ECREATE also validates the information used to initialize the SECS and result in a page fault or a general fault, if the information is not valid. A good example of this is to specify the enclave size as a power of 2. So if this condition is not satisfied while setting the SIZE field, then ECREATE results in general fault. Finally ECREATE initializes the enclaves INIT attribute to false. This is so that the enclave's code cannot be executed until the INIT attribute is set to true. This happens in the initialization step.

3.4.2 Loading

This step in the life cycle is when the code is loaded into the enclave. When the enclave's SECS is in the uninitialized state (set in the creation step), the system software uses the EADD instruction to load the initial code and data into the secure enclave.

EADD reads input data from a Page Information structure (PAGEINFO). The PAGEINFO[45] structure can only be used to communicate information in the SGX implementation. EADD validates its inputs before modifying the newly allocated EPC pages. So attempting to EADD an EPC page to an enclave in initialized state will again result in general fault. Similarly, attempting to EADD an EPC that is already allocated results in a page fault.

This step is also responsible for the measurement calculation(MRENCLAVE) that is used in the software attestation in Section 3.3.3.

3.4.3 Initialization

After *loading* an enclave in the *loading* step, where the data pages are loaded into the enclave, the system software uses a *Launch Enclave (LE)* to obtain a EINIT token structure. This token marks the enclave's SECS as initialized when passed to the EINIT instruction. The LE is a privileged enclave from Intel and is a prerequisite for the use of enclaves. The SGX API provided by Intel abstracts this intricate process. Since LE is a SGX enclave it has to be created, loaded and initialized as a normal enclave. Also the LE is cryptographically signed with a key that is hard-coded into the hardware. This makes EINIT initialize the LE without checking for a valid EINIT Token Structure.

After the initialization step is successfully completed, it sets the INIT attribute to true. This enables the software application to execute the enclave's code. Furthermore, once INIT is set to true, EADD cannot be invoked on that enclave anymore, so the system software must load all the

pages that make up the enclave's initial state before executing the EINIT instruction. This ensures the enclave's integrity guarantee.

3.4.4 Tear Down

Finally, once the enclave's purpose is completed, the EREMOVE instruction deallocates the EPC pages used by the enclave. EREMOVE marks an EPC page as available by setting the VALID field to 0. Also before freeing up the page, EREMOVE confirms that there is no logical processor executing the enclave code that has ownership of the page which is about to be destroyed.

An enclave is destroyed completely when the EPC page that hold the SECS (which is created in the first step) of the enclave is freed. EREMOVE also will not deallocate a SECS if it is referenced by any other enclave. This is to make sure that an enclave's SECS page can only be deallocated after all the enclave's pages have been removed.

As mentioned above, it is important to understand an enclave's lifecycle to avoid any memory leaks and make sure that any unreferenced or deallocated memory is removed and hence is free of any vulnerabilities.

3.5 Conclusion

This chapter extensively covered the security features of Intel SGX, namely memory design, data sealing and software attestation. This knowledge comes in handy while incorporating SGX into a legacy application or while developing a stand-alone SGX application. The two different software attestations and their respective steps for implementation were illustrated and explained with an example.

The second part of this chapter highlighted and explained the crypto guarantees and primitives that make SGX a viable security solution. The workings of each of these primitives were explained to get a different perspective of how the above-mentioned functionalities can be helpful with real-life threat vectors and scenarios. This part also explained the assumptions of these primitives, which make any security system vulnerable if not implemented properly.

This chapter concluded with the lifecycle of an SGX enclave, from initialization to teardown. This included all the associated sub-process and structures which aid in the development of an SGX application. It is important to know about the life cycle of an enclave to properly create

and discard enclaves without any data leaks or security vulnerabilities.

4 Methods and Implementation

4.1 Introduction

Software applications often need to work with private information, commonly financial information, encryption keys, health records etc. This sensitive data is intended only to be accessed by the designated recipient. In the context of this work, we refer to this as *application secrets*. The OS's job is to enforce security policy on the computer system so that the secrets are not unintentionally exposed to other users or applications. Applications often employ safeguards for their data, such as data encryption, to ensure that data sent to storage or over a network connection cannot be accessed by unauthorized third parties, even if the OS and hardware are compromised.

Despite these protections, there is still a significant vulnerability. While there are many safeguards in place to protect one application from another or the OS from an unauthorized or unprivileged user, an application has no protection from higher privileged processes like the kernel or the OS itself. So if a malware gained privileged access to the OS or the kernel, it has unrestricted access to all system resources and applications running in the system. This same problem reflects in a cloud based platform too. So a sophisticated malware which infiltrated the cloud system can target an application's protection schemes to extract encryption keys and even the secret data itself directly from memory.

Hence, we propose a *TEE-based application* to securely upload multiparty data-sets. The application works solely with Intel SGX and will require an SGX-enabled machine to run on and with the right configurations.

Section 4.2 of this chapter has the different use cases in which this software system can be implemented. Section 4.3 covers an abstract outline of the software system requirements and how the software system works. Section 4.4 has an detailed implementation of the application in a cancer diagnostics usecase and explains how multiple clinics can collaborate securely to optimize and improve their computational model. The application is designed to reduce the attack surface and to keep it to a minimum. This is explained in Section 4.5 along with the different classes of the software software system. Section 4.6 of this chapter focuses on the enclave components and the flow of application secrets among them.

4.2 Use Case Solutions

There are three use case scenarios (Section 2.2.2) studied for this work. These use cases are spread among the top most data sensitive industries and would require complete process transparency but data confidentiality. The cut throat competition among them makes it difficult to develop a robust solution for collaborative computing or data transfer among them or with a 3rd party. This work proposes a system to provide a general viable solution to each of these use cases.

Medical Industry

In this use case, a pharmaceutical company has a machine learning model that can predict the occurrence of a disease to a certain extent. The company can then use this result to change production of medicines, Personal Protective Equipment (PPE), vaccines, etc. The data comes from clinical trials, most commonly sponsored by the same pharmaceutical company. But the companies are also interested in having more data coming from clinics from a different part of the country or from a different part of the world. Now the problem here is that these clinics (independent or sponsored) are very cautious with patient data and therefore are very reluctant to share it outside their jurisdiction. But at the same time the clinics do not possess the required hardware nor the knowledge to train the model locally at their sites.

Solution: A confidential machine learning platform built on top Intel SGX can potentially solve this problem. The application enables the clinics to encrypt their data locally and then send it over to the secure machine learning platform. The model gets applied to the data and computations are performed to get accurate predictions. Finally, the results are encrypted and sent to all participating parties. This enclave-based system guarantees that the data and the model remain confidential and tamper resistant at all times with integrity checks in place at all critical points.

Hence, the pharmaceutical company is able to

1. provide guarantees to the clinics that no plaintext patient data ever leaves their jurisdictions to convince them to participate in the research.

2. establish a secure link throughout the community and train their model on all the combined data without actually having access to it.

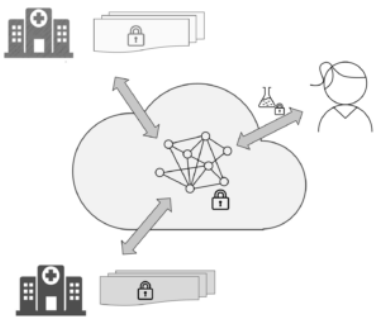


Figure 23: Use case: Medical Industry

Insurance Industry

This use case mostly involves data sharing. Trust-based data sharing is a vital component of oblivious computation systems. This use case shows that one component of this proposed work can be used to solve an important use case. The insurance industry is well known for its trust issues. A business interruption and property policies insurer needs to better understand the current state of an entire supply chain. The insurer has to increase their understanding of the risks, current value-in-port policies, or faster estimate damages in the case of disaster against which that particular business is insured. So a holistic view on the port of the goods or business like the location, content, value of all the relevant goods and transactions will aid the insurer to make proper decisions to release reserves and improve customer interactions. The current problem is that the required data is owned by the supply-chain participants. These participants are reluctant to share data as they are very sensitive and if a competitor got hold of their data, it would negatively impact their business. So if the communication among the supply chain participants and the insurers is not effective, it results in improper customer interactions.

Solution: An enclave-based system solves trust issues through the process of remote attestation. After the entities undergo remote attestation, the supply chain participants can share the cryptographic keys to decrypt their separately uploaded data-sets. The application can then be queried for the appropriate results. The database is designed for the enclave and is capable of storing the data sets. Queries can include the total value of goods in a port over time or distributions of time-in-port over all goods.

Hence the data remains confidential in the enclave and the insurer can provide data integrity

guarantees through processes like the MRENCLAVE calculations.

Banking Industry

Banks depend a lot on churn prediction and fraud detection to identify illegitimate transactions and customers. Churn prediction uses big data to detect customers who are likely to cancel their subscriptions, accounts and services. It is used by most of the large subscription businesses and banks to identify customers most at-risk of churning. If a good churn prediction model is in play, banks can have huge business savings and customer bases, even if implemented in small scale. Fraud detection[6] is a set of activities undertaken to prevent money or property from being obtained through false pretenses. In banking, fraud may include forging checks or using stolen credit cards. Banking and insurance industries use standard machine learning for churn prediction and fraud detection. As with all machine learning, the more data that is used for training, the higher the prediction score is. This means that the model performance highly correlates to the available data. Banks in general, not just small banks, often suffer from very few datapoints to build good, high performance models. In the case of smaller banks, which combine this issue with lack of data science infrastructure and expertise, this create a desire and need to collaborate among similar organizations.

Solution: Similar to the above use cases, the solution relies on the trust developed with an enclave-based application. For example, the machine learning algorithm for fraud detection runs as an enclave-based service in the cloud, which is ready to receive as many data points as possible. Using SGX capabilities like software attestation, the data is shared with the service after a required trust has been established. The service then runs the training and outputs the resulting model to all the participating organizations securely.

This allows all the participating organizations to collaboratively train machine learning models on confidential and sensitive data without any of these organizations having to trust each other. The organizations initially establish trust only with the platform and this is done in a very transparent manner. The massively increased amount of training data now results in very accurate prediction scores and high performance in detecting fraud. This, in turn, reduces the cost for the organizations.

[6] https://searchsecurity.techtarget.com/definition/fraud-detection

4.3 Process Outline

The design of the system requires the application code to be divided into two main code bases: The *Trusted Code Base* and the *Untrusted Code Base*.

1. **Trusted code base** is the enclave itself. The code base that resides in the trusted code is the code that has access to the application secrets. An application can have more than one trusted code base or, in other words, more than one enclave.

2. **Untrusted code base** is the rest of the application, its UI modules etc. In a TEE design, the OS, the kernel and the Virtual Machine Manger VMM are considered to be part of the untrusted code base.

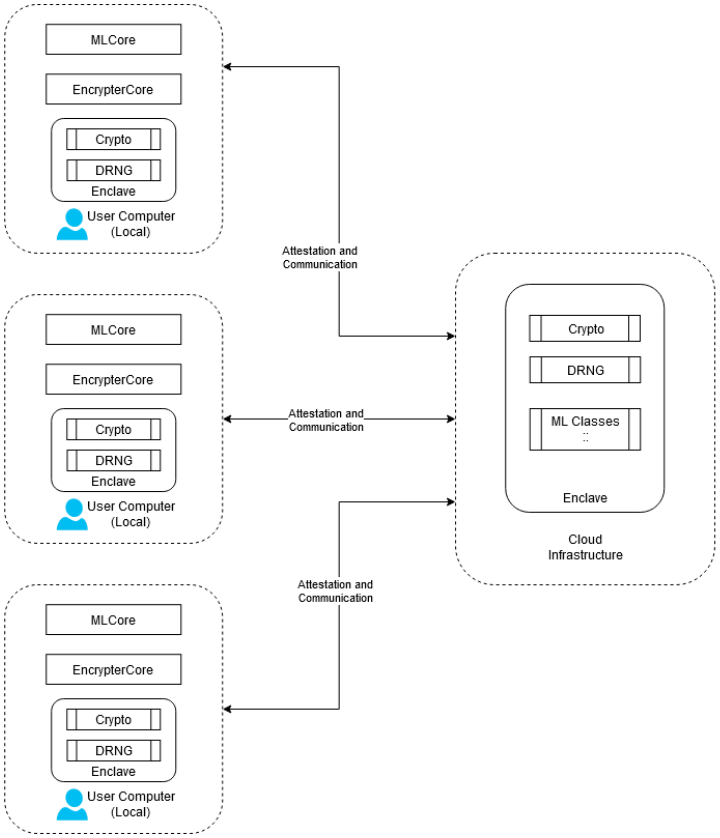


Figure 24: Architecture - Secure remote computation.

Intel SGX makes it possible to run a code on a remote system, typically a cloud provider such that the remote system cannot alter the program that is executed, which maintains integrity,

and also the program can process encrypted data that is decrypted only inside the CPUs (cloud) and therefore is inaccessible to the OS, VMM or the kernel. The system is designed to move the trust from the cloud provider to the application code and Intel SGX.

The illustration shown in figure 24 shows a top level view of implementation. Once a secure connection and trust is established through attestation, the user initiates sensitive data communication to the cloud. Since the machine learning code resides in the enclave, the data flow inside the enclave is in plaintext.

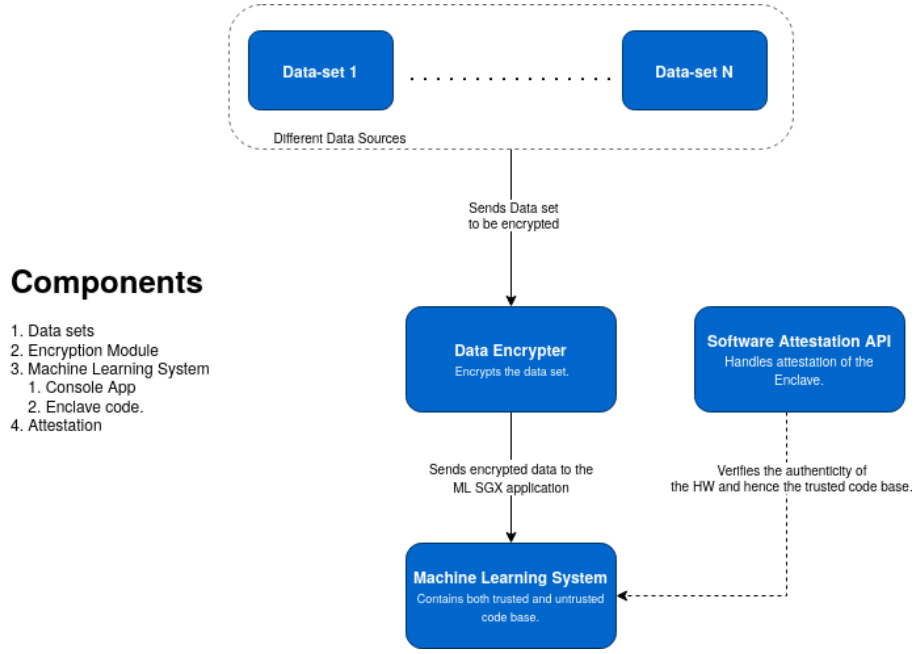


Figure 25: C1 Architecture

To preface the discussion of core software components, classes, and their design, an abstract view of the architecture is given in Figure 25. The systems contains 2 main modules: the *Encrypter Module* and the *Machine Learning module*. The Encrypter module is responsible for sealing the data using authenticated encryption. The Encrypter module resides completely in the user's system and is bound to the enclave used to seal the secrets.

The Machine Learning module resides in an SGX-enabled cloud system. This module is responsible for decrypting the application secrets and performing the necessary computations. The machine learning module is not exposed to the OS, the VMM or any third party. Before the data

is shared, the user confirms the identity of the ML module using SGX software attestation (Section 3.3.3). Depending on the use case and implementation, this attestation can be local or remote. For this work, since the cloud environment is simulated in the local system, a local attestation is done to authenticate the identity of the enclaves before secure data communication takes place.

The steps to use the application are as follows:

1. User registration is done and keys are generated using Windows Security Support Provider (SSP) API.
2. The user authenticates herself to the system using Windows Security Support Provider Interface (SSPI) and establishes a session. This session is timed and recorded by the local system.
3. The user encrypts the data set(s) using the encrypter module and the encrypted file(s) are saved locally.
4. The user uploads encrypted data sets to the machine learning module though the application UI.
5. The user trains on the all the data in the repository and is given the learning and test accuracy.

4.3.1 Application requirements

Some basic application requirements can help narrow down the scope of the work so that the focus can be solely on the Intel SGX integration rather than the finer details in design and development of the application.

Requirements and Design Decisions	1. Only runs on Intel SGX capable systems or Visual Studio 2017 with Intel SGX SDK. 2. Requires Microsoft Windows 10 64-bit for Intel SGX support. 3. Requires an Intel processor that supports Intel Data Protection Technology(DPT) with Secure Key 4. Must not rely on 3rd party libraries or utilities.

Table 2: Requirements and Design Decisions

Requirements and Design Decisions	1. C++ based Win32(WinAPI) Front end and Back end 2. Requires Microsoft Windows 10 64-bit for Intel SGX support. 3. Must not rely on 3rd party libraries or utilities.

Table 3: UI Requirements and Design Decisions

Table 2 shows the system requirements to run the proposed work. The 3rd requirement gives access to RDRAND instruction, which ensures access to a high quality entropy source. Table 3 shows the requirements of the UI module and the corresponding design decision impacts. The main reason for using Win32 for the application is to reduce usage of managed code, such as frameworks. Developing applications with managed code adds an extra layer of complexity and communication flows, which in turn has an impact on the attack surfaces.

4.4 Use case - Cancer Diagnostics

4.4.1 Introduction

The etiology of breast cancer involves several physiological, genetic, environmental factors. Estrogen metabolism as catechol estrogens were found to be the main reason for high incidence of breast cancer in women. Certain genetic traits were reported to induce high level of catechol estrogen production[49]. Thiamine, co factors of folate pathway like riboflavin and vitamin B6 were reported

to confer protection against breast cancer[50]. Many other researchers have explored these factors in detail and their possible association with breast cancer. A robust machine learning model can be useful in classifying patients with breast cancer by using the data collected and predicting whether a patient has the disease.

4.4.2 Diagnostics and Machine learning

Even before a person develops cancer, a machine learning based prediction model may provide insights into their susceptibility to the disease. For a long time it has been known that ANNs have diagnostic utility, with an early review concluding that machine learning methods are more effective than traditional statistical approaches, particularly for large data sets.

After a diagnostics has been made, ANNs can be used to improve treatment plans in better ways than a trained medical professional. 15 years ago, cancer prediction models were able to predict bladder cancer relapse with an accuracy rate of 88-95%. This level of accuracy is significantly more reliable than standard logistic regression techniques. In recent years, multiple start-ups and service providers began trending because of their research with the application of machine learning in cancer disease modeling by intelligently deciphering complex data sets for both disease recurrence and patient survival. The main limiting factor is the availability of patient sensitive data to increase the performance of these models. Since patient data is protected by strict privacy and security rules, the data is not easy to collect, share and distribute.

4.4.3 Data and Variables

In this work, a risk prediction model for breast cancer is used as an example to simulate a real life use case of the practicability of the proposed system. The model incorporates demographic data, dietary intake of folate and certain micro nutrients as the primary factors to be accessed. The rationale of this prediction model is to access the influence of lifestyle, genetics and micro nutrient intake. A demographic factor like age or BMI and a measure of of micronuritent like folates and B6 is taken as the input data. The genetic factor that is taken in consideration is Glutamate carboxypeptidase II (GCPII), which is predominantly expressed in the brain, intestinal mucosa and prostate cancer in the form of three splice variants i.e. N-acetylated-α-linked acidic dipeptidase (NAALADase), folyl poly-γ-glutamate carboxypeptidase (FGCP) and prostate specific membrane antigen (PSMA)

respectively. Its inhibition was found to confer protection against certain neurological disorders and also breast cancer[51].

The training data contains information about the patients genetics, demographics and nutrition along with information about which individual developed breast cancer. This is the data that is used to train the neural network. Here 75% of the data is used for training and 25% is used for testing. The test data is used to find out how good the network is operating. The output variable is whether the patient had developed breast cancer(Figure 26).

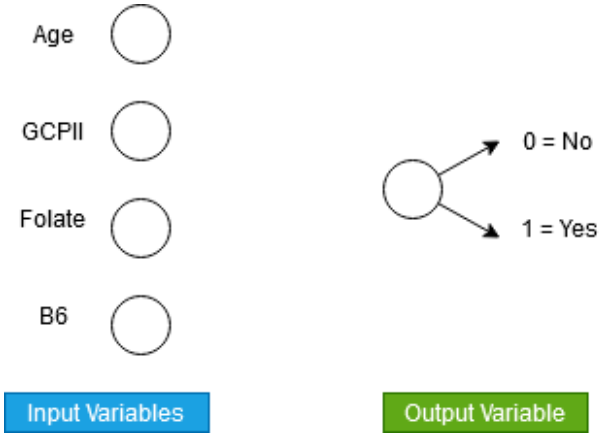


Figure 26: Input and Output Data Variables

4.4.4 Use Case Simulation and Working

The system aims to provide a collaborative computing platform by implementing an enclave based ANN to predict breast cancer. In this simulation, multiple clinics are planning to collaboratively share and engage in a machine learning system hosted by a 3rd party med-tech company. The med-tech company has an ANN model designed and developed in-house and runs in a secure enclave in the cloud. The ANN model is a propriety technology of the med-tech company and no details of the network design or implementation are disclosed to the participating clinics or the public. The company has its Intel EPID key so that the participants can verify the authenticity of the service.

The simulation shows how the participating clinics can share their data securely to the company's enclave based service and monitor the model's training and testing accuracy.

Encrypter

1. The participating clinics are required to download and install 2 applications: Encrypter Application and ML Application. The local system requirements to install and run the applications are given in Table 2.

2. **Encrypter Application**

   2.1. The clinics open the Encrypter application and authenticate themselves (Figure 27).

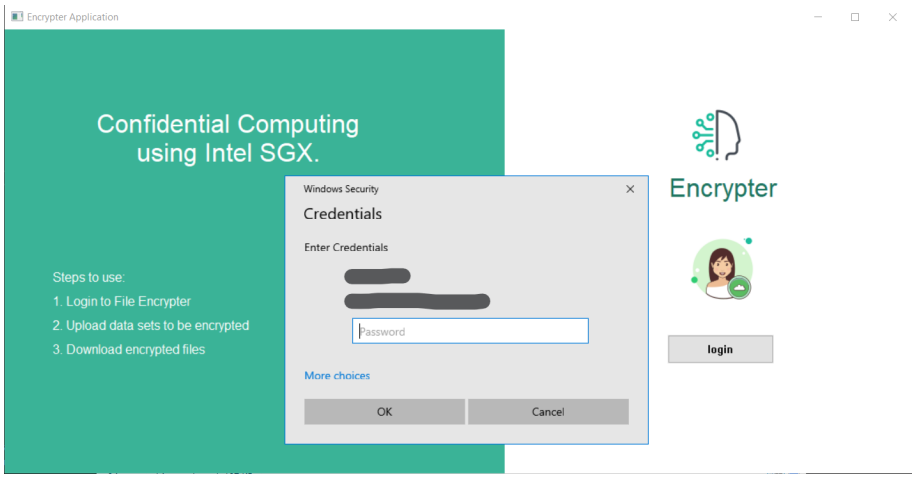


Figure 27: Encrypter Application - Login

The Login Interface through which the clinic (user) can authenticate themselves and initiate a session.

   2.2. The clinics encrypt the data set(s) by clicking on the "upload" button (Figure 28). The Windows file picker interface is used to select the file from the local system. Each encrypted data set is listed as a tile under "Encrypted Files". The encrypted file name is shown on the application output console box. The user can download the encrypted files to the local system by clicking on the respective tile (Figure 29).

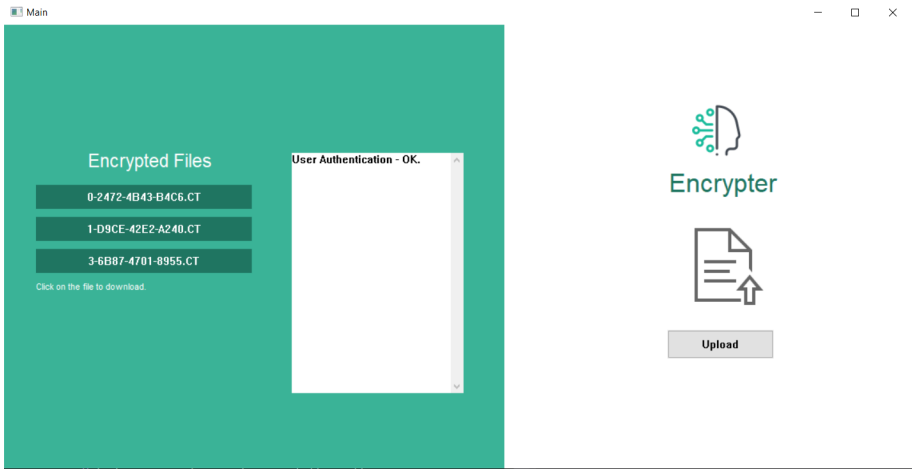


Figure 28: Encrypter Application - Main

This screen of the Encrypter App is used to upload the files that need to be encrypted. This interface also displays the files encrypted by the user. The user is able to download the encrypted data to the required destination on the local computer.

Figure 29: Encrypter Application - After encryption

A random file name is generated for every encrypted file. The new file name is displayed on the console and a new tile is added to the list of encrypted files.

3. **ML Application**

The ML application is the interface between the clinics and the secure cloud service run by the med-tech company. The application requires the clinics to upload the encrypted data to the platform.

3.1. The clinics open the ML application and authenticate themselves to the system (Figure 30).

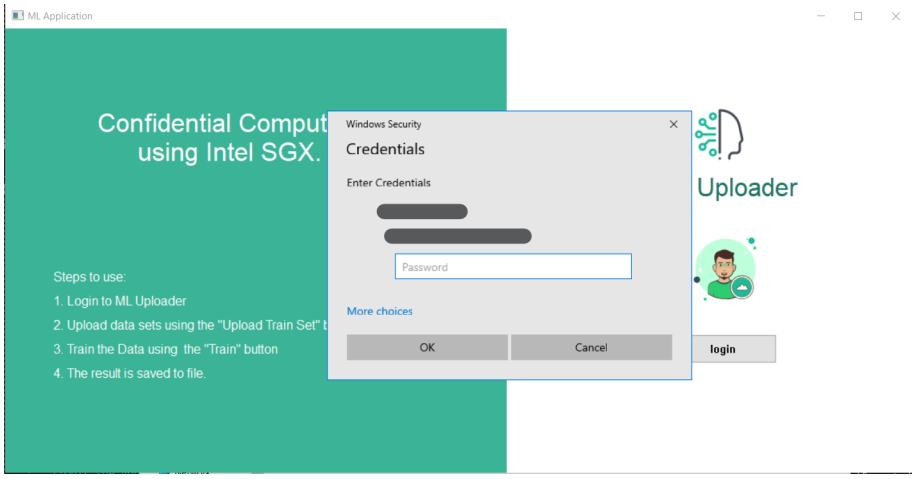


Figure 30: ML Application - Login

The Login Interface through which the clinic (user) can authenticate themselves and initiate a session.

3.2. After authentication, the clinic can either upload an encrypted data set or train the model and view its performance (Figure 31). If the clinic chooses to upload the encrypted data set as a part of the training data, she can click on the "Upload Train Set" button. The Windows file picker interface is used to select the file to be uploaded 32.

3.3. Finally, the clinic can train the network by clicking on the "Train" button (Figure 33).

The working of the system is illustrated in Figure 34.

Figure 31: ML Application - Main

After successful login, the next screen of the ML application displays a list of all encrypted data sets and functionalities to upload more encrypted data sets to the system and/or to train on the data.

Figure 32: ML Application - After File Upload

After clicking on the "Upload Train Set" button, the selected file is uploaded to the system. A new tile is added to the list of encrypted files in the system.

Figure 33: ML Application - Train Model

After the user clicks on the "Train" button, all the encrypted data sets are decrypted and the network is trained on them. The training and testing results are printed to a file and stored on the local computer.

4.4.5 Results

The network's accuracy increases with the number of participating clinics and their data sets. The patient data is secure and tamper resistant. The med-tech company is now able to provide a guarantee that no raw patient data is disclosed outside the clinic's jurisdiction and can provide accountability for the service. Different Hospitals or clinics do not have access to the data uploaded by the other participants but can access the trained model and results within their access rights. For simulation purposes, a simple ANN is used, which gives an accuracy of up to 95% with all the sample data sets.

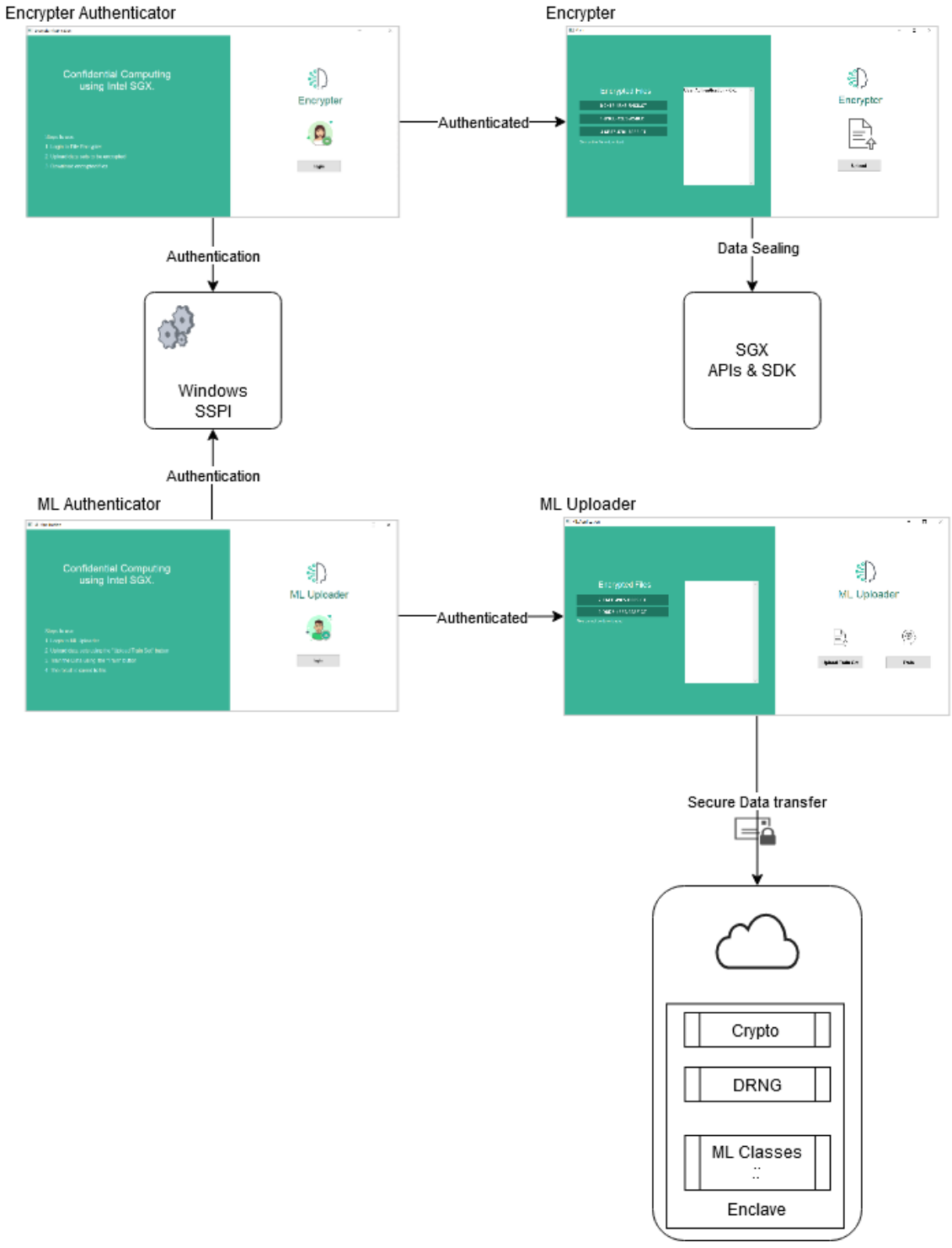


Figure 34: Application Workflow

4.5 Software design

The trusted component should be as small as possible, limited to the data that needs the most protection and those operations that must act directly on it. A large enclave with a complex interface doesn't just consume more protected memory: it also creates a larger attack surface.

Enclaves should also have minimal trusted-untrusted component interaction. While enclaves can leave the protected memory region and call functions in the untrusted component (through the use of a special instruction), limiting these dependencies will strengthen the enclave against attack.

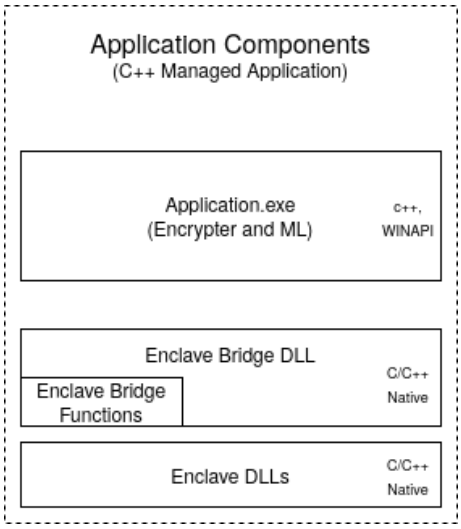


Figure 35: Components of the system.

There are two main parts of this application: trusted and untrusted code. The trusted code resides in the enclave DLLs as shown in the Figure 35. The enclave code can be C or C++ and is secured using Intel's SGX security guarantees. One main drawback of SGX is that enclaves cannot directly interact with the input/output modules of the system. This means that it is not possible for a user to directly communicate with the enclave code.

How to communicate communicate with the enclaves? This communication usually entails sharing of data, which can be application secrets, from the UI to the enclave. Even though this data can be encrypted, there are other security concerns like freshness or integrity. This communication can be done effectively using bridge functions, which are usually wrappers around the ECalls or

OCalls. The implementation of bridge functions depends solely on the application design and the technology stack used. If the application is built using managed code, these wrapper functions can bridge the gap between the enclave code which is written in C/C++ and the application. For this work, bridge functions are written even though they are redundant, to show how to robustly develop an Intel SGX application.

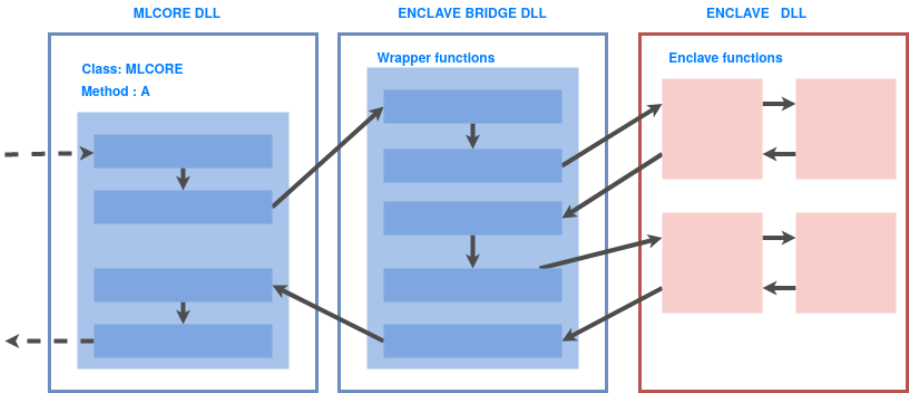


Figure 36: Execution Flow Diagram

The Figure 36 shows how an untrusted part of the application can communicate with the secure enclave code. The majority of the ECalls are simple wrappers around the class methods in the vault. Enclave bridge DLL is comprised of wrapper functions of the methods of the enclave classes. This is done because of the complication that ECalls, OCalls and bridge functions must be in native C code. Hence, once the enclave is launched there is a need to span the gap between C and C++ objects.

Figure 37 shows the class diagram of the system. The only communication path between the trusted and untrusted code is from the FileHandler Class to the Crypto Class. The encrypted data is decrypted once it enters the enclave and moves around as plain-text within the enclave. The Crypto methods are

1. Encrypter - The encrypter method reads the data and seals it using a local enclave. The sealing function used is AES-GCM which is described in Section 3.2.7. Data can be sealed using two modes: Seal by Enclave ID or Seal by Author. For the sake of simplicity, in this work, the data is sealed by using the enclave author mode.

2. Decrypter - This method resides as enclave code in the cloud as part of the ML module and

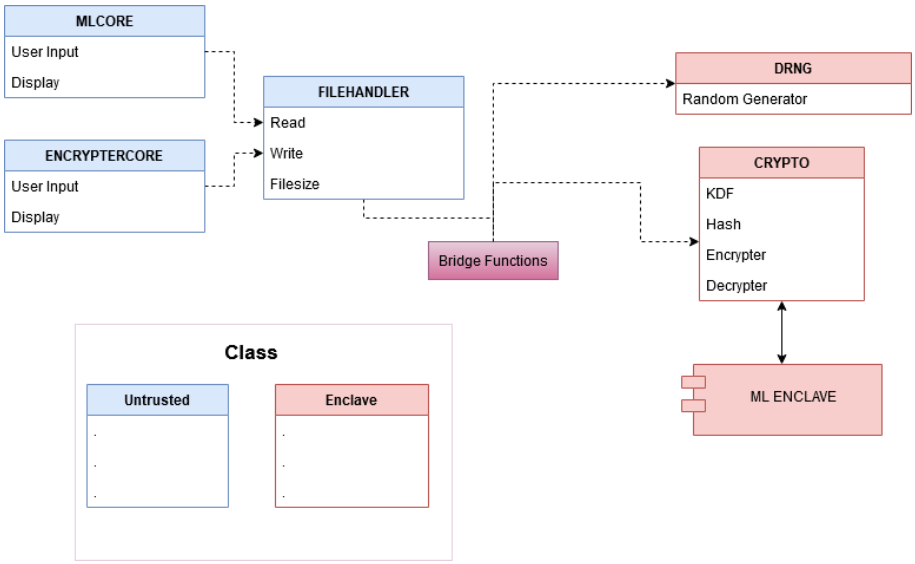


Figure 37: Class Diagram.

unseals the data passed to it. The decryption module can only unseal data after the encrypter enclave has been authenticated by the cloud enclave once per session using software attestation methods of Intel SGX.

3. KDF - The Key Derivation Function generates a 128 bit key that can be used for AES using SHA2.

4. Hash - Hashing method to generate secure cryptographic hashes.

Another class in the enclave code is the DRNG class. The DRNG is the interface to the on-chip digital random number generator, which uses the Intel Secure Key. There are SGX alternatives to the Intel Secure Key technology. The DRNG takes care of the RDRAND and RDSEED feature detection checks and instruction calls. The DRNG class[7] generates 512 pairs of 128-bit values and mix the intermediate values together using the CBC-MAC mode of AES to produce a single, 128-bit seed. The process is repeated to generate as many seeds as necessary.

The MLCore, EncrypterCore and FileHandler are the untrusted counterparts that don't reside in any enclave. These mainly consist of UI methods and file handling methods. The MLCore

[7]https://software.intel.com/content/www/us/en/develop/articles/intel-digital-random-number-generator-drng-software-implementation-guide.html

and EncrypterCore classes contain interfaces like secure login and get inputs from the user, like the dataset uploader. These classes interact with the filehandler in the untrusted code base.

4.5.1 Crypto Module

The crypto module of the application contains both trusted and untrusted code bases. This module requires the user to Login, access Windows APIs and provide enclave functions like data sealing, KDF, and so on. The crypto module resides completely on the local computer of the user with Intel SGX support. Figure 38 shows the communication and interaction between various components inside this module.

A registered user is required to authenticate himself to the system using Windows SSPI and hence initiates a session. The user's hash is retrieved by the crypto module to be further used to generate a UserKey. This UserKey is used to encrypt all the FileKeys under this user. A user is allowed to encrypt as many files as he wishes. Each of these FileKeys have to be persisted along with the encrypted message to be transferred through an insecure medium. Hence the FileKeys are encrypted with the UserKeys for stronger authentication and accountability.

Once a user is authenticated, the interface requests that the files be encrypted. The user inputs all the files to be encrypted. The files are then encrypted using AES-GCM method provided by Intel SGX. AES-GCM is an authenticated encryption method which requires a mandatory authentication tag that is used in the decryption process. So an encrypted file is comprised of the ciphertext, the encrypted FileKey and the authentication tag. The KDF requires a random salt as an input. The crypto module uses the random generator method of the DRNG class to generate the necessary salt for the KDF. Since the crypto module resides on the local computer of the user, the data sealing is done using the Enclave Author which is defined with the help of the user hash.

The crypto module allows the user to save the encrypted file (.CT format) anywhere on the computer. The user can open the file with any text editor. But any change to the ciphertext will be detected by the authenticated encryption algorithm and will result in decryption failure. Any replay attacks are detected using this system.

On the other end, the decryption method parses the .CT file for ciphertext, the encrypted FileKey and the authentication tag. The encrypted FileKey is first decrypted using the UserKey. The ciphertext is then authenticated for its integrity and then decrypted by the AES decryption

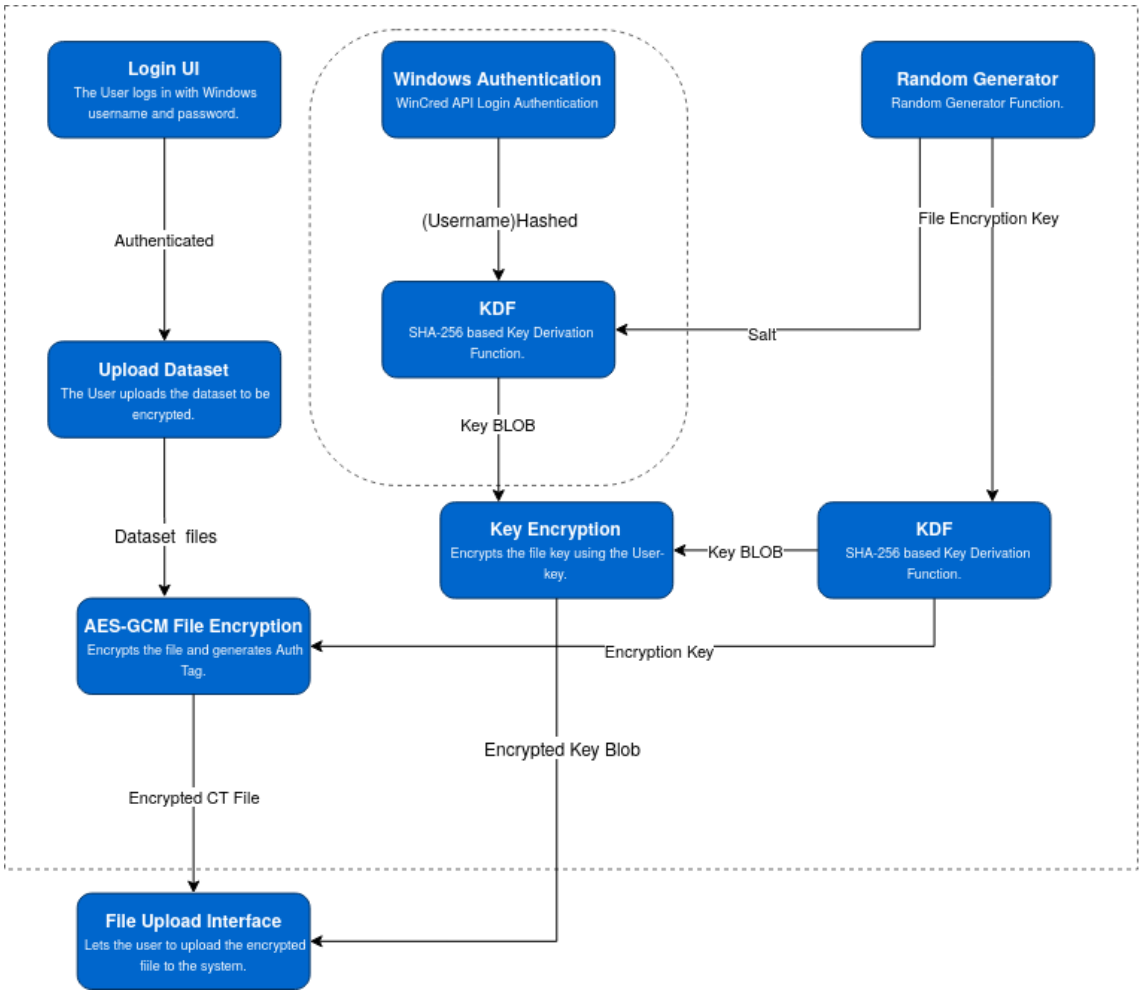


Figure 38: C2 - Crypto Module

algorithm.

The plaintext data after decryption doesn't leave the enclave and is hence secure from any attacks from malicious application in the cloud or the VMM, OS etc.

4.5.2 Machine Learning Module

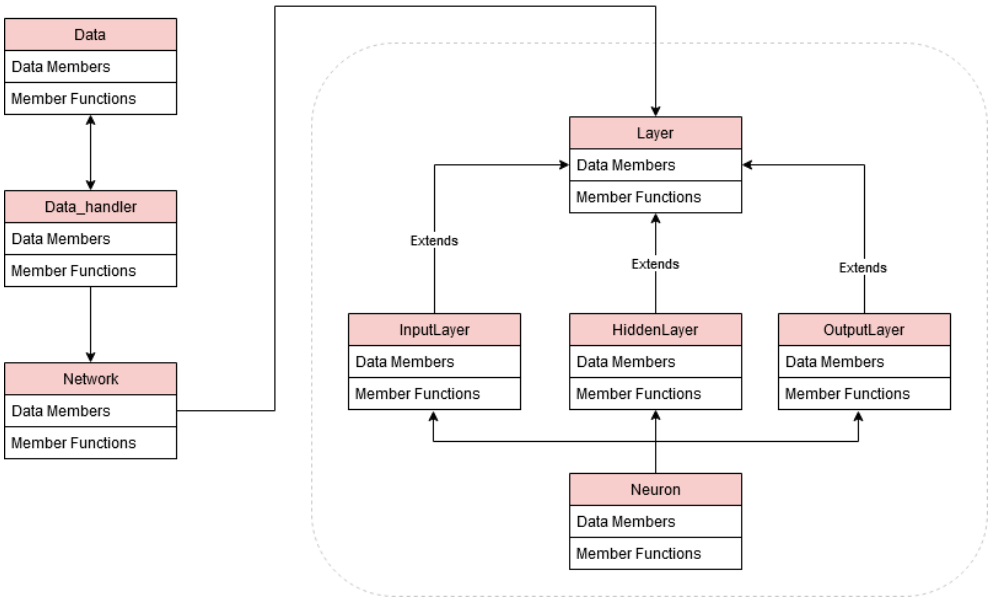


Figure 39: Machine Learning Enclave Code - Class Diagram

The ML module is comprised of classes that handle the plaintext data, classes for different layers and classes for individual neurons and networks of neurons. The ML architecture is a classic neural network setup with 3 different layers: input, hidden and output layer. The ML module requires the data to be in plaintext as opposed to homomorphic schemes. The plaintext data resides only in the enclave and is protected from attacks from the untrusted part of the cloud. No instance of the plaintext data is passed outside the enclave in any form.

Section 2.2.1 explains feed-forward neural networks with back propagation used in this work. The setup has 50 hidden layers and trains for 1000 epochs. The algorithm reads in CSV files.

4.6 Enclave Design

The enclave code is the most important part of the application design as it is the trusted part of the application. The goal is to minimize the number of components inside the enclave to reduce the attack surface.

In the user end, the enclave is needed to seal the data safely. Hence two classes resides here. The Crypto class and the DRNG class. The enclave in the remote server(cloud) has all 3 components namely, Crypto, DRNG and machine learning classes(Section 4.5.2). These classes together make the enclave code, so it is important that they are implemented effectively and are free of bugs.

4.6.1 Enclave Logistics

The first step in designing the enclave is working out a system to manage the enclave itself. The enclave must be launched properly and the resulting enclave ID must be provided to the ECalls. This setup is ideal if this is transparent to the upper layers of the application. One easy solution is to use global variables in the Enclave Bridge DLL to hold enclave information. This design decision comes with a restriction: unless there is a locking mechanism of some sort, only one thread can be active in the enclave at a time. But for simple applications single pipelined threads are more than enough.

What happens if an enclave is lost due to a power event or a bug that causes it to crash? For that, we check the return value of the ECall: it indicates the success or failure of the ECall operation itself, not of the function being called in the enclave[45]. Any result other than SGX_SUCCESS indicates that the program did not successfully enter the enclave and the requested function did not run.

4.6.2 Enclave Definition Language

When building an enclave using Intel SGX, the interface has to be defined to the enclave in the *Enclave Definition Language (EDL)*. An enclave's bridge functions, both its ECalls and OCalls, are prototyped in its EDL file. When the project is built Intel SGX, SDK parses the EDL file and generates a series of proxy functions. Each ECall and OCall gets a pair of proxy functions: a trusted half and an untrusted half. The trusted functions go into EnclaveProject_t.h and EnclaveProjct_t.c and are included in the Autogenerated Files folder of the enclave project. The untrusted proxies go

into EnclaveProject_u.h and EnclaveProject_u.c and are placed in the Autogenerated Files folder of the project that will be interfacing with the enclave. The proxy functions are responsible for

1. Marshaling data into and out of the enclave.
2. Placing the return value of the real ECall or OCall in an address referenced by a pointer parameter.
3. Returning the success or failure of the ECall or OCall itself as an sgx_status_t value(Success or error).

Its general structure of an EDL file is as follows:

```
enclave{
  // Include files
  // Import other edl files
  // Data structure declarations to be used as parameters
  // of the function prototypes in edl

  trusted {
  // Include file if any. It will be inserted in the
  // trusted header file (enclave_t.h)
  // Trusted function prototypes (ECalls)
  };

  untrusted {
  // Include file if any. It will be inserted in the
  // untrusted header file (enclave_u.h)
  // Untrusted function prototypes (OCalls)
  };
};
```

5 Security Considerations

TEEs in general are not a new technology. Hundreds of millions of mobile devices worldwide rely on TEEs for protection of security-critical applications like DRM and OS components like *Android Keystore*. A TEE that is widely used in android devices is the *ARM TrustZone*. Unlike the Intel SGX, the ARM TrustZone uses a different architecture and implementation ideologies. But at the same time, TEEs have been successfully attacked with highly damaging impacts across various platforms. Unfortunately, these attacks have been possible not just by the presence of security flaws in TEE systems, but also the flaws and bugs in the application that runs in them. This section focuses on the security aspects of Intel SGX. It is to be noted that most of these points are applicable to other TEEs as well.

This section explains the threat model, security aspects, both positive and negative, of this system. Section 5.3 explains the external authentication system (windows SSPI) used in the application and its associated security considerations. Finally, the possibility of side channel attacks in which a malicious user can get in to the system or gain access to the application security is explored in explained in Section 5.4.

5.1 Threat Model

For this work we assume the standard SGX threat model. The adversary is in complete control of privileged software, in particular,the hypervisor and the operating system. They are able to spawn and stop processes at any point, set their affinity and modify it at runtime. The adversary can also read and write to any memory region except the enclave memory, map any virtual page to any physical one and dynamically re-map pages. The attacker also managed to have control of the cloud and any communication from the participating users and the cloud.

5.2 Attack surfaces

To preface the security aspects of this work, it is important to know how Intel Software Guard Extensions Technology helps secure data.

The Figure 40 shows the attack surface for a traditional application and for a enclave application.

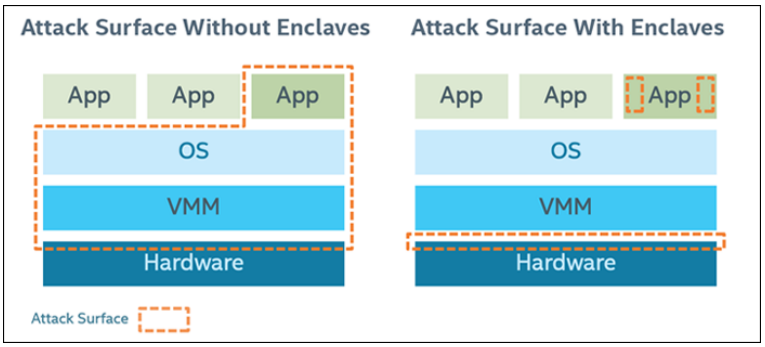


Figure 40: Attack surface with and without Intel SGX.

Intel SGX offers the following protections from known hardware and software attacks:

1. Reading and writing to an enclave memory cannot be done outside the enclave.
2. Many exploits use debugger vulnerabilities or misplaced debug code to break an application. Intel SGX can only be debugged by the Intel SGX Debugger. Production enclaves cannot be debugged by any software or hardware debuggers.
3. Since applications need Ecalls and Ocalls to enter and leave the enclave, classic function calls, jumps, register manipulations or stack manipulations will not work in an enclave. The only way to call enclave functions is through a new instruction which performs several protection checks.
4. Enclave memory is encrypted using industry standard authenticated encryption, which protects against replay attacks. This also means that trying to tap on to the memory directly or using the memory chip in another CPU will only yield encrypted data.
5. The memory encryption key is randomly changed between every power cycle. This key is then stored within the CPU and in not accessible by any other software.
6. Data stored within enclaves can only be accessed by code that shares the enclave.

In addition to the intrinsic security limitations of SGX documented by Intel, attackers can easily compromise the security of an SGX-based application by exploiting the bugs in the enclave software such as classical memory corruption or concurrency issues (SGX will not magically secure insecure software). Unsafe SGX programming by the enclave author such as mishandling in-enclave

pointers, or leaking secret data through function calls outside the enclave might result in failure of the system.

Focusing on the development process, an insecure development environment can make the enclave application vulnerable to exploits. If the OS is compromised when developing the enclave, then integrity of that enclave and any code within it is also compromised and should not be trusted.

Despite Intel's suggestion to use 3rd party libraries, doing so can be a very serious vulnerability in the application. If the 3rd party library is compromised then so is the enclave code. Bugs in the Intel trusted libraries (libc or crypto library, for example) are also potential issues, but these are addressed regularly by Intel.

Even though the CPU code cannot interact directly with the enclave, CPU bugs, in the hardware logic or in the microcode, can be used to attack the enclave.

5.3 Windows SSPI

The Windows SSPI is a common interface between transport-level applications, such as Microsoft Remote Procedure Call (RPC), and security providers, such as Windows Distributed Security. SSPI allows a transport application to call one of several security providers to obtain an authenticated connection. In this work, the Windows SSPI is called by the win32 application to authenticate the user and initiate a secure session. Security packages support security protocols such as Kerberos authentication and the Microsoft LAN Manager.

SSPI allows an application to use various security models available on a computer or network without changing the interface to the security system. The SSPI doesn't create or handle logon credentials. This function is a privileged operation and is handled by the OS.

There are various authentication packages[8] provided by Microsoft, including the following:

1. *Credential Security Support Provider*
2. *Microsoft Negotiate*
3. *Microsoft NTLM*
4. *Microsoft Kerberos*

[8] https://docs.microsoft.com/en-us/windows/win32/secauthn/ssp-packages-provided-by-microsoft

5. *Microsoft Digest SSP*

6. *Secure Channel*

The *Microsoft Credential SSP* or *CredSSP* is used in this work. CredSSP lets an application delegate the user's credentials from the client to the target server for remote authentication. CredSSP provides an encrypted Transport Layer Security Protocol channel. The client is authenticated over the encrypted channel by using the *Simple and Protected Negotiate (SPNEGO)* protocol with either Microsoft Kerberos or Microsoft NTLM. After the client and server are authenticated, the client passes the user's credentials to the server. The credentials are doubly encrypted under the SPNEGO and TLS session keys. CredSSP supports password-based logon as well as smart card logon.

Coupled with the Windows SSPI, the Winlogon module is used for an interactive logon. The Winlogon module uses Windows 10 credential providers as the primary mechanism for user authentication. The credential providers are the only method for users to prove their identity, which is required for logon and other system authentication services. There are several ways to logon to the application such as passwords, PIN, smartcard, fingerprint, Face and Iris recognition, which are called system credential providers. This API also includes support for 3rd party credential providers.

To understand the security considerations of the authentication system as a whole, it is important to understand how Windows credential management works. It is a process in which the OS receives the credentials from the application, service or user and secures that information for future presentation to the authentication target. The credentials used in authentication are digitally verifiable documents that associate the user's identity to some form of proof of authenticity, such as a certificate, a password, or a PIN.

By default the Windows credentials are stored on the local computer and are validated against the Security Accounts Manger(SAM) database on the local computer as well. Credentials are requested and collected through user input on the logon user interface or coded through the application programming interface (API) to be presented to the authenticating target.

The credentials, for example the password, is stored as a hash in the local system. Using the Windows credential manager increases the attack surface in the local system significantly. An attacker can pull credentials from many different areas of the system. Once the attacker gains access to a regular endpoint computer, the attacker can look for the credentials in :

1. *WDigest*
2. *SAM*
3. *Local Security Authority (LSA) secrets*
4. *Kerberos System*

Even though the credentials stored here are encrypted or hashed, the attacker can attempt to pass them to the authentication server. If the server simply compares the encrypted credentials (hashed passwords), then it will match and the attacker will gain access. This vulnerability, called "Pass the Hash"[48], where the attacker exploits the password hash saved in the cache memory or by stealing (credential dumping) from the above listed places. This exploit lets the attacker gain administrator-level permissions for a user's machine. On the other hand, Windows credential manager is secure as long as the users themselves are trusted.

There are many ways to mitigate and defend against credential dumping. If databases like SAM and LSA are monitored and logged to keep a look out for unexpected connections from IP addresses not assigned to known domain controllers, the risk of malicious activities is reduced.

What happens if an attacker already has all the credentials or if the endpoint computer cannot be trusted? How can one defend the integrity and confidentiality of the system? The solution is *Multi-Factor Authentication (MFA)*. MFA can serve as a second source of authentication system to verify user identity before granting access. Another way to defend against an attacker form moving laterally through a network and gaining access to a machine is to implement a zero-trust strategy.

5.4 Side Channel Attacks

Side Channel Attacks are those that target the vulnerability of a computer system and gain information from the implementation, rather from than the weakness, of the software or algorithm. Timing information, power consumption, electromagnetic leaks or even sound can provide an extra source of information that can be exploited. Since Intel SGX is a secure hardware-based TEE, it is important to analyze potential side channel exploits that might compromise the confidential computing system.

The latest side channel exploits and research works are explored in Section 2.2.5. Common side channel attacks that might exploit enclaves are cache attacks, timing attacks and software-initiated fault attacks. Even though software initiated attacks are a class of side channel attacks,

the work done in section 2.2.5 classifies software attacks outside of side-channel attacks. Spectre attacks [36] come under software initiated attacks which exploit speculative execution of enclave code. The cache timing attacks on enclaves by [39] shows how a malware can run inside an enclave as enclave code and compromise other enclaves in the same hardware by mounting a DRAM attack.

6 Conclusion

Security and privacy are the most significant concerns for both enterprises and consumers with rise of cloud technologies. The need for secure data and data computation is the primary motivation of this thesis. This work aims to minimize the data exposure only to the intended authorized cloud subsystem, while maintaining integrity and confidentiality of both the data and its computation. Confidential computing aims to set a security standard to protect the data-in-use. It furthermore strives to provide better user authority and transparency that is especially fitting to public clouds.

This work proposes a system of confidential computing using Intel SGX as a trusted environment and to study the practical implications of such a system. The work explores the design aspects of the system and also studies the security considerations of each of the components.

The state of the art in Chapter 2 reviews the technologies in the field of neural network based machine learning and confidential machine learning and concludes with an exploration of TEEs and attacks on TEEs. The TEE chosen for this work is Intel's SGX technology, which is the latest and one of the more secure options among those on the market. A thorough background study on Intel SGX is conducted in Chapter 3. Chapter 4 is comprised of the implementation details of the practical work, including the solutions offered to use cases and the entire software design of both trusted and untrusted parts of the system. The threat model and the security considerations of the system components are explained in Chapter 5.

This study concludes that, like any technology, Intel SGX is vulnerable to attacks. Intel claims to have complete security from such attacks, but many studies show chinks in the SGX technology. Enclave applications are definitely safer than traditional applications and since Intel SGX is a trust based system, is has, over the years, definitely earned this trust. It is important not to include numerous 3rd party libraries and plugins, which have an adverse effect on the whole attacks surface of the system. Once the trust is partitioned and delegated to many components, the probability of exploits increases drastically. Responsible software design and development is key for minimizing risks of system penetration and unauthorised access.

6.1 Future Works

The security of an enclave based software system relies mainly on the code and implementation. The Intel SGX SDK has many protocols in place to securely develop and deploy enclave applications. Intel promises to release regular updates with the SDK which will make it easier to develop enclave applications. Many container based implementations are also being developed by third party researchers and developers to add an additional layer of security to the enclave technology. One way to increase the popularity of Intel SGX is to enable enclave programming in other major programming languages like Rust, Python, Swift etc. There are existing frameworks like Fortanix EDP[9] using Rust for enclave application development but none of these are officially supported by Intel.

The two main areas of confidential computing: FHE and enclave technologies, are still evolving and will be able to provide a stronger guarantee on the security of the computations and data, in the future. Further works are being done on increasing the computational performance of an enclave by securely leveraging the GPU by creating a secure environment on GPUs. Increasing the EPCs are a good way to write larger applications which is important in many real-life use-cases. We believe more use cases like blockchain will be enclave based. Secure computation is also on demand and will continue to evolve because the need to incorporate inputs from multiple data owners like (Amazon, Facebook, Google etc) for a single machine learning problem like product recommendation.

[9] https://edp.fortanix.com/

References

[1] MITRE Corporation,Common vulnerabilities and exposures (CVE) details: The ultimate security vulnerability datasource. browse vulnerabilities by date, 2019. Accessed on 10/10/2019. [Online]. Available: https://www.cvedetails.com/browse-by-date.php.

[2] Rafael Pereira Pires, "Distributed systems and trusted execution environments:Trade-offs and challenges", arXiv:2001.09670v3 [cs.CR] 11 Mar 2020, Available: https://arxiv.org/pdf/2001.09670.pdf

[3] D. Snyder, Intel unleashes next-gen enthusiast desktop pc platform at gamescom, 2015. Available: https://blogs.intel.com/technology/2015/08/6th-gen-gamescom/.

[4] Redhat Corporation, Current Trusted Execution Environment landscape,December 2, 2019, Accessed on 07/05/2020. [Online]. Available: https://next.redhat.com/2019/12/02/current-trusted-execution-environment-landscape/

[5] GENTRY, C. Fully homomorphic encryption using ideal lattices. In ACM Symposium on Theory of Computing (STOC) (2009).

[6] Thore Graepel, Kristin Lauter, and Michael Naehrig1, ML Confidential: Machine Learning on Encrypted Data, Microsoft Research.

[7] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from Ring-LWE and security for key dependent messages. In Phillip Rogaway, editor, CRYPTO, volume 6841 of Lecture Notes in Computer Science, pages 505–524. Springer, 2011.

[8] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, Innovations in Theoretical Computer Science – ITCS 2012, pages 309–325. ACM, 2012.

[9] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Phong Q. Nguyen and David Pointcheval, editors, Public Key Cryptography, volume 6056 of Lecture Notes in Computer Science, pages 420–443. Springer, 2010.

[10] Sanjiv Ranjan Das, Dataa Science Theories, Models, Algorithms and Analytics. Available: http://algo.scu.edu/sanjivdas/

[11] Kevin Gurney, An introduction to neural network, University of Sheffield, ISBN: 1-85728-673-1, UCL Press, 1997.

[12] Andy Thomas, An Introduction to Neural Networks for Beginners. Accessed on : 12.05.2020 Available : https://adventuresinmachinelearning.com/wp-content/uploads/2017/07/An-introduction-to-neural-networks-for-beginners.pdf

[13] Saad ALBAWI , Tareq Abed MOHAMMED, Saad AL-ZAWI, Understanding of a Convolutional Neural Network, ICET2017, Antalya, Turkey

[14] TCG: Trusted computing group, Trusted platform modules (TPM) part 1, design principles, 2011. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/TPM-Main-Part-1-Design-Principles_v1.2 _rev116_01032011.pdf.

[15] ARM security technology - building a secure system using TrustZone technology, ARM Limited, 2009. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492 C_trustzone_security_whitepaper.pdf.

[16] S. Pinto and N. Santos, "Demystifying arm TrustZone: A comprehensive survey", ACM Computing Surveys, vol. 51, no. 6, 130:1–130:36, 2019. doi: 10.1145/3291047

[17] AMD, EXTENDING SECURE ENCRYPTED VIRTUALIZATION WITH SEV-ES, KVM FORUM -2018 Available : https://events19.linuxfoundation.org/wp-content/uploads/2017/12/Extending-Secure-Encrypted-Virtualization-with-SEV-ES-Thomas-Lendacky-AMD.pdf

[18] AMD, Developer Central, Available : https://developer.amd.com/sev/

[19] David Kaplan, Jeremy Powell, Tom Woller, AMD Memory Encryption. Available : https://developer.amd.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf

[20] Yoongu Kim,Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, Onur Mutlu, Carnegie Mellon University and Intel Labs, Flipping Bits in Memory Without Accessing Them:An Experimental Study of DRAM Disturbance Errors. Available: https://people.inf.ethz.ch/omutlu/pub/dram-row-hammer_isca14.pdf

[21] intel Developer Zone, Accessed on 15.05.2020, Available : https://software.intel.com/en-us/forums/intel-software-guard-extensions-intel-sgx/topic/737218

[22] MERKEL, D. Docker: Light weight Linux Containers for Consistent Development and Deployment. Linux Journal (Mar.2014).

[23] KUBERNETES. Available : http://kubernetes.io, 2016

[24] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark L Stillwell, David Goltzsche, David Eyers, Rudiger Kapitza, Peter Pietzuch, and Christof Fetzer, SCONE: Secure Linux Containers with Intel SGX.

[25] M.Abadi, P.Barham, J.Chen, Z.Chen, A.Davis, J.Dean, M.Devin, S.Ghemawat, G.Irving, M.Isard, et al. Tensorflow: A system for large-scale machine learning. In OSDI,volume 16, pages 265–283, 2016.

[26] Roland Kunkel,Do Le Quoc,Franz Gregor, Sergei Arnautov, Pramod Bhatotia, Christof Fetzer, TensorSCONE: A Secure Tensor Flow Framework using Intel SGX, arXiv:1902.04413v1 [cs.CR] 12 Feb 2019.

[27] Zhongshu Gu et al., YerbaBuena: Securing Deep Learning Inference Data via Enclave-based Ternary Model Partitioning, IBM Research, arXiv:1807.00969v2 [cs.CR] 16 May 2019

[28] H. Shacham, "The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86)", in Proceedings of the 14th ACM Conference on Computer and Communications Security, ser. CCS '07, Alexandria, Virginia, USA, 2007, pp. 552–561. doi: 10.1145/1315245.1315313

[29] T. Bletsch, X. Jiang, V. W. Freeh and Z. Liang, "Jump-oriented programming: A new class of code-reuse attack", in Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, ser. ASIACCS '11, Hong Kong, China: ACM, 2011, pp. 30–40. doi: 10.1145/1966913.1966919. [Online]. Available: http://doi.acm.org/10.1145/1966913.1966919

[30] A. Baumann, M. Peinado and G. Hunt, "Shielding applications from an untrusted cloud with haven", in 11th USENIX Symposium on Operating Systems Design and Implementa-

tion (OSDI 14), Broomfield, CO: USENIX Association, 2014, pp. 267–283. [Online]. Available: https://www.usenix.org/conference/osdi14/technicalsessions/presentation/baumann

[31] C. che Tsai, D. E. Porter and M. Vij, "Graphene-SGX: A practical library OS for unmodified applications on SGX", in 2017 USENIX Annual Technical Conference (USENIX ATC 17), Santa Clara, CA: USENIX Association, 2017, pp. 645–658. [Online]. Available: https://www.usenix.org/conference/atc17/technical-sessions/presentation/ tsai.

[32] Daniel Moghimi, Berk Sunar, Thomas Eisenbarth, and Nadia Heninger, TPM-FAIL:TPM meets Timing and Lattice Attacks, 29th USENIX Security Symposium (USENIX Security 20), Available: https://www.usenix.org/conference/usenixsecurity20/presentation/moghimi

[33] Eyal Ronen, Robert Gillham, Daniel Genkin, Adi Shamir, David Wong, and Yuval Yarom. The 9 Lives of Bleichenbacher's CAT: New Cache ATtacks on TLS Implementations. In IEEE Symposium on Security and Privacy, 2019.

[34] Kit Murdock, David Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens, Plundervolt: Software-based Fault Injection Attacks against Intel SGX, Proceedings of the 41st IEEE Symposium on Security and Privacy (S&P'20), Available : https://plundervolt.com/

[35] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx, Foreshadow - Extracting Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution, 27th USENIX Security Symposium. August 15–17, 2018 • Baltimore, MD, USA ISBN 978-1-939133-04-5

[36] KOCHER, P., GENKIN, D., GRUSS, D., HAAS, W., HAMBURG, M., LIPP, M., MANGARD, S., PRESCHER, T., SCHWARZ, M., AND YAROM, Y. Spectre attacks: Exploiting speculative execution. arXiv preprint arXiv:1801.01203 (2018).

[37] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, Ten H. Lai, SgxPectreAttacks: Stealing Intel Secrets from SGX Enclaves via Speculative Execution, arXiv:1802.09085v3 [cs.CR] 3 Jun 2018

[38] Johannes Götzfried at al. Cache Attacks on Intel SGX, EuroSec'17: Proceedings of the 10th European Workshop on Systems Security, April 2017, Available: https://doi.org/10.1145/3065913.3065915

[39] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice and Stefan Mangard, Malware Guard Extension: abusing Intel SGX to conceal cache attacks. Available : https://doi.org/10.1186/s42400-019-0042-y

[40] Kocher PC, Timing Attacks on Implementations of Diffe-Hellman, RSA, DSS, and Other Systems.In:Crypto'96. Available: https://doi.org/10.1007/3-54068697-5_9

[41] D. Page, Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel, University of Bristol Technical Report CSTR-02-003, Sumbitted to TISSEC, Available: https://eprint.iacr.org/2002/169

[42] Bernstein DJ, Cache-timing attacks on AES. Technical report. Department of Mathematics, Statistics, and Computer Science, University of Illinois, Chicago. DOI: 10.1007/978-3-662-44709-3_5

[43] COLIN PERCIVAL, CACHE MISSING FOR FUN AND PROFIT, Available: https://papers.freebsd.org/2005/cperciva-cache_missing/

[44] Intel, Intel® Software Guard Extensions, Developer Guide, Available: https://01.org/sites/default/files/documentation/intel_sgx_sdk_developer_reference_for_linux_os_pdf.pdf

[45] Victor Costan and Srinivas Devadas, Intel SGX Explained, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

[46] NIST, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, NIST Special Publication 800-38D November, 2007

[47] D. Eastlake and P. Jones. RFC 3174: US Secure Hash Algorithm 1 (SHA1). Internet RFCs, 2001.

[48] https://www.blackhat.com/docs/us-15/materials/us-15-Moore-Defeating%20Pass-the-Hash-Separation-Of-Powers-wp.pdf

[49] Naushad, S. M., Ramaiah, M. J., Pavithrakumari, M., Jayapriya, J., Hussain, T., Alrokayan, S. A., ... Kutala, V. K. (2016), Artificial neural network-based exploration of gene-nutrient interactions in folate and xenobiotic metabolic pathways that modulate susceptibility to breast cancer, Gene, 580(2), 159-168. doi:10.1016/j.gene.2016.01.023

[50] Cancarini, I., Krogh, V., Agnoli, C., Grioni, S., Matullo, G., Pala, V., Pedraglio, S., Contiero, P.,Riva, C., Muti, P., Sieri, S., 2015.Micronutrients involved in one-carbon metabolismand risk of breast cancer subtypes. PLoS One 10 (9), e0138318.

[51] Divyya S, Naushad SM, Addlagatta A, et al. Paradoxical role of C1561T glutamate carboxypeptidase II (GCPII) genetic polymorphism in altering disease susceptibility. Gene. 2012;497(2):273-279. doi:10.1016/j.gene.2012.01.055