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

Contents

Acknowledgements **2**

List of Figures **3**

List of Abbreviations **4**

1 Introduction **5**

1.1 Definitions of key terminologies 6

1.2 Structure of the Thesis 7

2 State of the Art **8**

2.1 Introduction 8

2.2 Background Information 8

2.2.1 Machine Learning 8

2.2.2 Confidential and Oblivious Machine Learning 13

2.2.3 TEE Technologies 15

2.3 TPMs 15

2.3.1 Attacks on TPMs 19

2.4 Research Questions 19

2.5 Conclusion 19

3 Methods and Implemenation **20**

3.1 Process Outline 20

4 Security Considerations **20**

5 Conclusion **20**

Acknowledgements

List of Figures

1 Simple example of neural network [11] 9
2 A Simple Node 10
3 Effect of adjusting weights. 10
4 Three layer neural network 11
5 Simple Back Propagation 13
6 Use-case: Hospital scenario 14
7 TPM components [2] 16
8 ARM TrustZone components [2] 17
9 SEV Security Model [18] 18
10 AMD secure encrypted virtualisation (SEV) components [2] 18

Acronyms

FHE Fully Homomorphic Encryption. 14

FLD Fisher's Linear Discriminant. 15

LDA Linear Discriminent Analysis. 9

LMC Linear Means Classifier. 15

TPM Trusted Platform Modules. 6

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 performs access control by separating software roles into multiple user modes. Privileged system software like hyper-visors and operating systems control that manage machine resources and other vital components of the system, the user developed applications rely not only on the correctness of the system software, ie, they are free of bugs that are exploitable, but also on the honesty of system softwares. The correctness of the system software is vastly exploited by hackers as this is very hard to verify. The main reason for this is 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 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 are many, like cost of operations and maintenance, 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, still one should wonder about the other tenants using the same platforms and are not malicious.

The solution is safe data processing and execution environments inside untrusted or compromised computers. They can be used for confidential data processing in public clouds where the

need of protected encrypted code and data from malicious administrators and hackers.

As software systems have such large code bases, the attack surfaces are proportionally large. TEEs invert this logic. 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 bases, TCBs. As the attack surfaces are reduced and 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, but 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 software like OSs and kernels are compromised. Given the numerous 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 a 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. Difficulties in implementation of the software system.
3. What benefit 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 part is inspired by the background study performed in the field and is an abstract view of the different technologies. First and foremost, we have to distinguish TEEs, TPMs and similar technologies like HSMs.

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 Hardware Security

Modules (HSMs). However TEEs are for fundamentally different compared to these hardwares. TEEs can be implemented along with these hardware.

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 least immediately evident of the tampering. TPMs are not designed to provide general computational capacity. They do provide some basic computational capabilities like generation of random keys, encrypt 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.

A 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 OS does not handle encryption 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 CPU and motherboard, accessed via PCI bus, network, or similar.

SGX terminologies

1.2 Structure of the Thesis

This part will be filled at the end.

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, multiple hospitals can pool in patient data to train a model that will aid in diagnosing a disease. Machine learning stands on the principle of 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 [5] are powerful cryptographic tools that can be used for privacy preserving machine learning. But many research in the field reports large runtime overloads, which limits 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). Then, an overview of popular attacks on TEEs are studied.

2.2 Background Information

2.2.1 Machine Learning

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, and 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, ho uses machine learning to predict movie revenues.

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 Discriminent Analysis (LDA) where the ML system is given a set of labelled data(ex. vector representations of digits along with the label). On the other hand unsupervised learning includes reorganizing and enhancing the input data to find some patters in it and to place a structure on the unlabelled data. Cluster Analysis is a trivial example of unsupervised leaning. Here the system tries to find a pattern among the individual data points and tries to cluster them with respect to 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 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 powers 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].

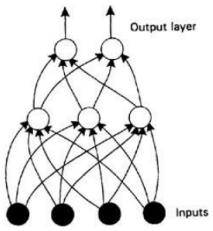

Figure 1: Simple example of neural network [11]

"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

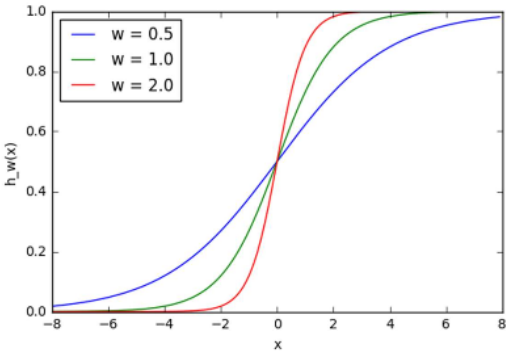

Figure 3: Effect of adjusting weights.

We can observe(Figure 2) 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.

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

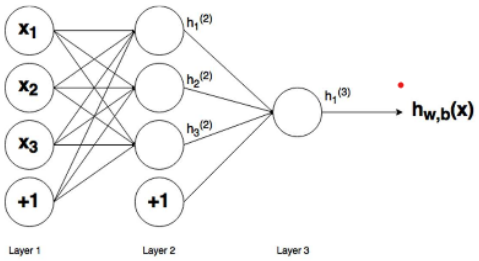

Figure 4: Three layer neural network

Feed-forward Pass

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

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

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

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

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

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.

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 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 of a change in weight, 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 output 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 networks best connection to the cost function. So how much a node in the hidden layer contributes to $\delta_i^{(n_l)}$ and how? The answer for how it contributes is, via the weight $w_{ij}^{(2)}$.

As can be observed from Fig.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:

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

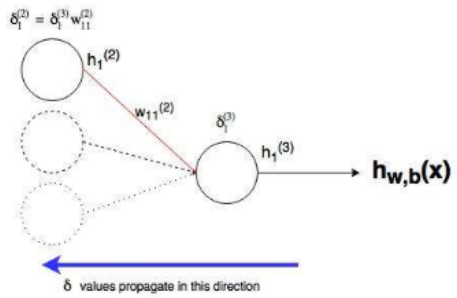

Figure 5: Simple Back Propagation

In the case of multiple output nodes, the weighted sum of all communicated errors are considered to calculate $\delta_j^{(l)}$. Hence 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].

2.2.2 Confidential and Oblivious Machine Learning

Machine Learning consists of two stages, the training stage and the classification stage. Either one of 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, who have been authorized by the data owners. Remote devices, sensors etc can be a good example of content providers. Using the same example of a hospital system, the patient, the hospital or some large company of lab technicians can be the data owners. The Cloud Service may be run by a third party, a partner company, 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 etc which

directly send data to the cloud where it is processed.

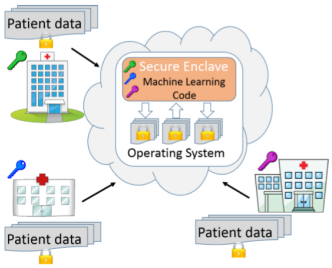

Figure 6: Use-case: Hospital scenario

This kind of design is an ideal scenario for a trust based multiparty computation system. Thore et al's rationale for proposing these protocols is that there are 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 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.

When outsourcing computation to a 3rd party service makes sense but confidentiality of the data is an issue. The way to preserve data confidentiality is to encrypt the data before uploading it to the cloud. This may limit the usage of the data. There are many recent advances in cryptography allows 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 called a Fully Homomorphic Encryption (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 have different TPM implementations focusing on Intel SGX.

2.2.3 TEE Technologies

2.3 TPMs

TEEs and TPMs are explained in Section 1.1. TPMs are independent and are tamper-resistance co-processor. 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[1] of a TPM is illustrated in 7.

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 to use 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: how TPM chips are wired to main processors, what are the privileges users must hold to get access to such chips and whether adversaries might have physical access[2]. Security attacks on TPMs are analyzed in section 2.3.1.

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

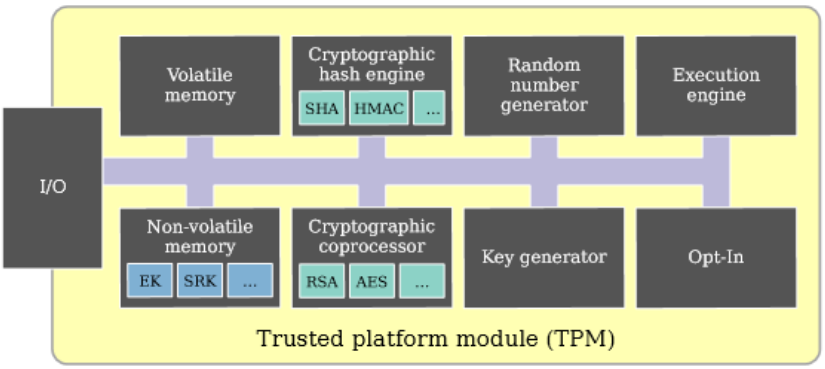

Figure 7: TPM components [2]

ARM TrustZone

ARM TrustZone[14] 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 world). In this architecture, each of the processor or core, 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 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[14], [15].

ARM has not specified any Remote attestation mechanism or data sealing directly but many implementations assumes 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

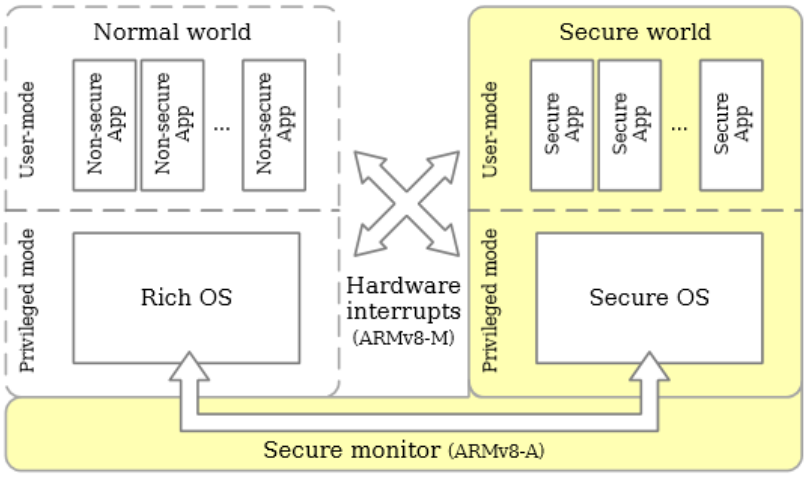

Figure 8: ARM TrustZone components [2]

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[15]

AMD secure encrypted virtualisation (SEV)

AMD's SEV is designed to protect data in use by virtual machines(VMs). It uses hardware encryption engine embedded in the memory controller (MC) which performs cryptographic operations. This method provides minimum performance impact and it requires no changes for 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.

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. But 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[16].

SEV involves the hypervisor in many ways. The main use of the hypervisor in SEV is that, it

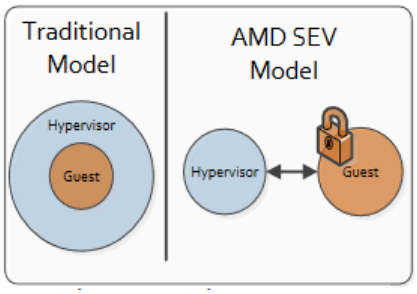

Figure 9: SEV Security Model [18]

manage 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 is responsible for key generation and storage. ASIDs are also used in cache pages to tag data for faster page hits.

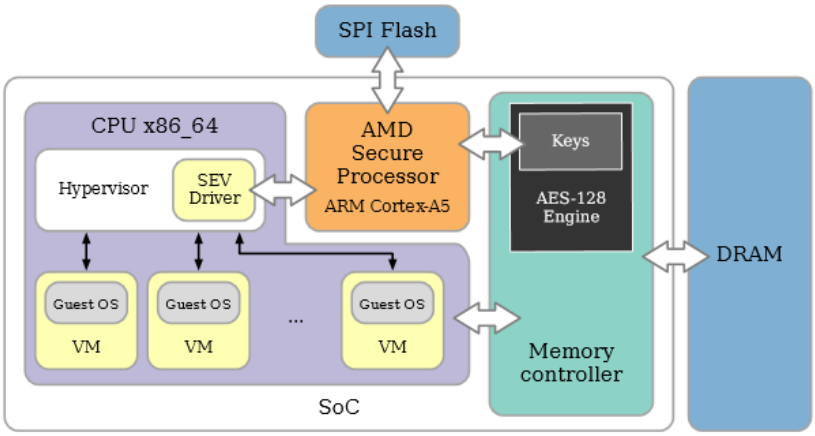

Figure 10: AMD secure encrypted virtualisation (SEV) components [2]

AMD also introduced an updated version of SEV called AMD Secure Encrypted Virtualization-Encrypted State (SEV-ES)[17]. 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 were 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[19]) and rollback attacks, i.e., when a previous state (even if encrypted and signed) is replayed back and considered genuine.

SCONE TensorSCONE YerbaBuena-IBM

2.3.1 Attacks on TPMs

Foreshadow: Plundervolt $\mathrm{SGX}_{C}acheAttack2.pdfSgxPectreAttacksMalwareGuardExtension : abusingIntelSGXtoco$

2.4 Research Questions

W

2.5 Conclusion

3 Methods and Implemenation

3.1 Process Outline

1. Key Generation
2. Encryption and Upload of Training Data
3. Training
4. Classification

4 Security Considerations

5 Conclusion

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] 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.

[14] 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.

[15] 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

[16] 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

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

[18] 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

[19] 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