Improving the Security of KMS on a Cloud Platform Using Trusted Hardware
Improving the Security of KMS on a Cloud Platform Using Trusted Hardware

For the past few years, the demand for cloud computing has increased rapidly. Users outsource data processing and storage of their private data to cloud systems. As the IoT industry is booming, cloud computing not only addresses the hardware and software restrictions of individual devices but also provides flexibility in resource allocation. According to the advantages, cloud computing plays an important role in the technology industry. However, the risk of data leakage and sensitive data exposed has raised when users outsource their data to a third party.

Currently, most cryptography based security techniques pay attention to the secret while in the application, at rest or in transit. With respect to the insider attacks, the sensitive data is in danger to be attacked by compromised devices without being noticed. In order to prevent insider threats, Hardware Security Module (HSM) provides a secure cryptographic solution to protect the data in an isolated space. However, compared with a software-based solution, it is costly and lacks the scalability. According to that, in this thesis, we apply a software-based technology, such as Intel Software Guard Extensions (Intel SGX) technology, to tackle the insider and outside threats towards the system.

The main idea of the research in this thesis is to utilize the Intel SGX technology in a key management service (KMS) in the cloud system to protect the sensitive data. The sensitive data inside the KMS is only processed within SGX enclaves, and implementing corresponding encryption functions within enclaves is also part of the thesis. In addition, the thesis analyses the performance implications of this solution. Moreover, we deploy the KMS with Intel SGX technology in a Kubernetes Cluster environment, in order to accomplish the high availability of the cloud system.

| Keywords: | Intel Software Guard Extension, Kubernetes, Hardware Security Module, Key Management Service |
| Language: | English |
Acknowledgements

This work was partly carried out in the D4Value project, funded by Business Finland.

I want to first thank my supervisory team: Vesa Hirvisalo and Jimmy Kjällman. Working with you was certainly a great learning experience. Vesa always gave me inspiring feedback and challenged me to think more in depth. I always felt motivated and driven after the meetings. Jimmy, my company mentor, he always supported and encouraged me to explore the topic in my own original way. Moreover, he always linked me to the right people in the company and helped me find the support and resources that I needed. This thesis will not be finished without him.

I would also like to thank all my colleague from NormadicLab. I really enjoyed our lunch and dance session, it help me get through the last mile in my master degree. I especially want to thank Miika, thank you introduce swing dance to me, it fulfills my life in my spare time.

Last but not least, I would like to thank my family and all my friends. Working on thesis is a great but sometimes stressful journey. Thanks for your love and mental support that pushed me through the tough times to finally get across the finish line.

Espoo, November 19, 2018

Shan Kuan
## Abbreviations and Acronyms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL</td>
<td>Access Control List</td>
</tr>
<tr>
<td>AEAD</td>
<td>Authenticated Encryption with Additional Data</td>
</tr>
<tr>
<td>AES</td>
<td>Advanced Encryption Standard</td>
</tr>
<tr>
<td>AESNI</td>
<td>Intel Advanced Encryption Standard New Instructions</td>
</tr>
<tr>
<td>AMD SEV</td>
<td>AMD Secure Encrypted Virtualization</td>
</tr>
<tr>
<td>AWS</td>
<td>Amazon Web Services</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>CTR</td>
<td>Counter Mode</td>
</tr>
<tr>
<td>DLL</td>
<td>Dynamic Link Library</td>
</tr>
<tr>
<td>DHKE</td>
<td>Diffie-Hellman Key Exchange</td>
</tr>
<tr>
<td>EC2</td>
<td>Elastic Compute Cloud</td>
</tr>
<tr>
<td>EDL</td>
<td>Enclave Definition Language</td>
</tr>
<tr>
<td>EPC</td>
<td>Enclave Page Cache</td>
</tr>
<tr>
<td>EPID</td>
<td>Enhanced Privacy ID</td>
</tr>
<tr>
<td>FaaS</td>
<td>Function as a Service</td>
</tr>
<tr>
<td>GCM</td>
<td>Galois Counter Mode</td>
</tr>
<tr>
<td>GMAC</td>
<td>Galois Message Authentication Code Mode</td>
</tr>
<tr>
<td>GF</td>
<td>Galois Field</td>
</tr>
<tr>
<td>HSM</td>
<td>Hardware Security Module</td>
</tr>
<tr>
<td>IaaS</td>
<td>Infrastructure as a Service</td>
</tr>
<tr>
<td>IAS</td>
<td>Intel Attestation Service</td>
</tr>
<tr>
<td>Intel IPP</td>
<td>Intel Integrated Performance Primitives</td>
</tr>
<tr>
<td>Intel SGX</td>
<td>Intel Software Guard Extensions</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>ISV</td>
<td>Independent Software Vendor</td>
</tr>
<tr>
<td>IT</td>
<td>Information Technology</td>
</tr>
<tr>
<td>IV</td>
<td>Initialization Vector</td>
</tr>
<tr>
<td>KMS</td>
<td>Key Management Service</td>
</tr>
<tr>
<td>MAC</td>
<td>Message Authentication Code</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>MitM</td>
<td>Man in the Middle Attack</td>
</tr>
<tr>
<td>NSS</td>
<td>Network Security Services</td>
</tr>
<tr>
<td>OS</td>
<td>Operation System</td>
</tr>
<tr>
<td>PaaS</td>
<td>Platform as a Service</td>
</tr>
<tr>
<td>PSW</td>
<td>Platform Software</td>
</tr>
<tr>
<td>QE</td>
<td>Quoting Enclave</td>
</tr>
<tr>
<td>REST-API</td>
<td>Representational State Transfer-Application Programming Interface</td>
</tr>
<tr>
<td>SaaS</td>
<td>Software as a Service</td>
</tr>
<tr>
<td>SPID</td>
<td>Service Provider Identity Document</td>
</tr>
<tr>
<td>SK</td>
<td>Session Key</td>
</tr>
<tr>
<td>SWIOTLB</td>
<td>Software Input Output Translation Lookaside Buffer</td>
</tr>
<tr>
<td>TCB</td>
<td>trusted computing base</td>
</tr>
<tr>
<td>TEE</td>
<td>Trusted Execution Environment</td>
</tr>
<tr>
<td>TLS</td>
<td>Transport Layer Security</td>
</tr>
<tr>
<td>TPM</td>
<td>Trusted Platform Modules</td>
</tr>
<tr>
<td>TXT</td>
<td>Trusted Execution Technology</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
</tbody>
</table>
# Contents

Abbreviations and Acronyms

1 Introduction
   1.1 Problem Overview ........................................... 10
   1.1.1 Outside Attack ............................................. 11
   1.1.2 Inside Attack ................................................ 11
   1.2 Approach Overview ........................................... 12
   1.3 Research Goal and Contribution ............................ 12
   1.4 Structure of the Thesis ...................................... 12

2 Background
   2.1 Trusted Hardware - Trusted Execution Environment ..... 14
      2.1.1 Intel Software Guard Extensions (Intel SGX) .......... 15
         2.1.1.1 Enclave .............................................. 15
         2.1.1.2 Sealing ............................................... 17
         2.1.1.3 Attestation ........................................... 18
         2.1.1.4 The Lifecycle of Intel SGX Enclave ................. 19
      2.1.2 ARM Trustzone .......................................... 20
      2.1.2.1 OP-TEE ............................................... 21
   2.1.3 AMD SEV .................................................. 21
   2.2 Kubernetes .................................................. 22
      2.2.1 Cluster Architecture ..................................... 23
      2.2.2 Etcd Storage Backend ................................... 24
   2.3 Key Management Service ..................................... 24
      2.3.1 OpenStack Barbican ...................................... 24
      2.3.2 Intel SGX Based Crypto Plugin of Barbican - BarbiiE 25
      2.3.3 Vault ..................................................... 27
   2.4 Cryptography ................................................. 31
      2.4.1 AES-GCM .................................................. 31
         2.4.1.1 Advanced Encryption Standard - AES ............. 31
         2.4.1.2 Counter Mode - CTR ................................ 32
2.4.1.3  Galois Message Authentication Code Mode - GMAC ..................................... 33
2.4.1.4  Galois Counter Mode - GCM .................................................. 34
2.4.2  Intel Advanced Encryption Standard New Instructions
       - AESNI ................................................................. 35

3  Adversary Model and Requirements .................................................. 36
   3.1  Adversary Model ................................................................. 36
   3.1.1  Goals and Motivations ................................................... 37
   3.1.2  Capabilities ................................................................. 37
   3.1.3  Related Attacks - Security Vulnerability .............................. 38
   3.1.4  Attack Scenario ............................................................ 38
   3.2  Performance Requirement .................................................... 39
   3.3  Security Requirement .......................................................... 40
   3.4  Deployability Requirement .................................................. 41
   3.5  Confidentiality, Integrity, and Availability (CIA) Triad ................. 41

4  Design ................................................................. 42
   4.1  Design Overview ............................................................... 42
   4.2  Components ................................................................. 43
       4.2.1  Local/Remote Attestation ............................................. 43
       4.2.2  Retrieve Encryption Key ............................................... 45
       4.2.3  Enclave ID Registry Server ........................................... 47
       4.2.4  Encryption/Decryption in the Enclave .............................. 48
   4.3  Scenario ................................................................. 49
       4.3.1  Client Stores Data to Server ....................................... 49
       4.3.2  Client Retrieves Data From Server .................................. 50

5  Implementation .................................................. 52
   5.1  Prerequisites ............................................................... 52
       5.1.1  Component Implementation .......................................... 53
       5.1.2  Local/Remote Attestation ............................................. 53
           5.1.2.1  Local Attestation ............................................... 53
           5.1.2.2  Remote Attestation ............................................. 54
       5.1.3  Retrieved Encryption Key ............................................ 54
       5.1.4  Encrypted/Decrypted in The Enclave ................................ 55
   5.2  Integration Overview ................................................... 56
       5.2.1  Embedded Development with C and Go ............................... 56
       5.2.2  System Integration .................................................. 57
6 Evaluation

6.1 Security Analysis .................................................. 59
  6.1.1 Offline Guessing Attack ..................................... 59
  6.1.2 Confidentiality and Integrity of the Sensitive Data ................. 60
  6.1.3 Minimizing the TCB .......................................... 60
  6.1.4 Other Issues .................................................. 60
  6.1.5 Conclusion ..................................................... 61

6.2 Performance Analysis .............................................. 61
  6.2.1 AES Cryptography - Throughput ............................. 61
  6.2.2 AES Cryptography - Latency ................................ 65
  6.2.3 Vault SGX vs Vault - Latency ............................... 67
  6.2.4 Other Issues .................................................. 67
  6.2.5 Conclusion ..................................................... 68

7 Conclusions .......................................................... 69
Chapter 1

Introduction

The evolution of cloud computing can be viewed as a step in the developing process of the IT industry. In the early 1960s [33], the computer scientist John McCarthy first introduced the time-sharing concept for the companies to rent the permission to use the resources on the mainframe [24], as the mainframe was not affordable to all the companies in general. Therefore, as the size of the mainframe was too huge, the companies preferred to share it rather than own it. According to that, the concept not only induces the computer as a service but also triggers the development of the internet.

For the next few decades, the usage of the personal computer is rising, and it takes over the time-sharing service of the mainframe. Meanwhile, the idea of the virtualization software has been launched, which makes the possibility of multiple users to log in and run a completely different operating system on the same computer simultaneously.

In 1997, Prof. Ramnath Chellappa provides the definition of the cloud computing - "Computing paradigm where the boundaries of computing will be determined by economic rationale rather than technical limits alone." [6]. At this time, the stock and technology industry suffers the severe collapse of the dot-com bubble. In the period, the market lost confidence to invest the money in the technology industry, it causes the amount of capital demanded to start a firm has plunged. However, thanks to the concept of Cloud Computing, it reduces the barrier of the startup company to operate the new business, and then to outsources the computer system and storage in the service vendor. Such as Amazon AWS, which introduces its Elastic Compute Cloud (EC2) in 2016 [40], it is the pioneer to provide the cloud service in public.

Thus, the concept of the IaaS (Infrastructure as a Service), PaaS (Platform as a Service), SaaS (Software as a Service) and FaaS (Function as a Service) has been rising in the IT market. Increasingly, IT companies join this technology race and then to divide the cloud computing market, which is
currently dominated by IBM, Google, Amazon, and Microsoft. The market includes all target customers, which vary from an enterprise to any individual user. The booming of the cloud computing market can be considered as another breakthrough in the technology industry in addition to the personal computer and the handheld device.

The user obtains huge benefits from cloud computing, for instance, the small business or freelancer can get significant benefit from cost saving in IT consumption, as they do not need to purchase the hardware equipment for data storage and the power consumption from the standby devices. Furthermore, the cloud also provides high availability, reliability, and manageability to the customers. It enhances the flexible working space and reduces the obstacle which can aid the user to improve the performance of the work [43]. However, despite the many advantages, the cloud comes with some drawbacks as well.

According to the characteristic of cloud computing, it is a never shut down system, which increases the risk of vulnerabilities which would be exploited by an adversary.

Furthermore, the cloud system supports multiple tenants to log in on the same devices. It also increases the risk of data leakage between the tenants or the person who has the privilege to monitor all the activity on the cloud platform [39].

This thesis utilizes the Trusted Execution Environment (TEE) concept which isolates the sensitive data of the main processor. The concept of TEE can prevent the attacks not only from the outside of the system but also the issue of insider attacks. Moreover, this study runs the TEE on a cloud platform to protect the sensitive data that users store in the storage of the cloud-based platform.

1.1 Problem Overview

In the past, users stored the data in the local devices, such as a hard disks, as the networking had a speed and bandwidth restriction at that time. Therefore, the market did not support the individual user to outsource the data storage, as it was too expensive and inconvenient for the individual user to purchase the outsourcing fee compared to the disc storage.

Currently, due to the popularity of the individual devices, people spend more time on the internet. According to the research in Adults’ Media Use and Attitudes Report by Ofcom in 2018, people spend 24 hours a week on

\footnote{https://www.ofcom.org.uk/research-and-data/media-literacy-research/adults/adults-media-use-and-attitudes}
the internet, it is two times more than one decade ago. According to that, people create more data and have a higher demand for processing speed. For this reason, if individual hardware and software cannot support the user in demands, it causes the user to outsourcing the data processing remotely. Furthermore, with the rapid progress of the internet, it drives the booming of the cloud computing industry consequently. However, it also raises the risk of the adversarial attacks and data leakage risks. In the following, we list two well-known adversarial attacks when people store sensitive data from the local devices to the cloud system remotely.

1.1.1 Outside Attack

Tampering happens when an attacker intercepts a message between two parties. However, the two parties are not aware that the message has been intercepted. The attacker can add, for instance, virus or malware to the message or edit the payload and then retransmit the message, or extract the sensitive data from the message.

In order to prevent, such as the Man in the middle attack (MitM) or Eavesdropping, Transport Layer Security (TLS) is typically used for the data transmitted in the network. TLS encrypts the data before the transmission, and the intended receiver can decrypt the received message with a private key. Although TLS addresses the MitM attack using the certificate authority system, it still at risk to expose the sensitive data to the fake server or client, as the client and service are not attested to each other before the secret sharing.

1.1.2 Inside Attack

If the adversary compromises the cloud system, or the maintainer of the cloud system steals the sensitive data, the cloud system is vulnerable to the insider threats. For instance, if the attacker has authorized to access the system, it can coordinate with malware from the outside to compromise the system. Therefore, it increases the risk of the external attacks on compromised devices.

Regarding the insider attacks, Hardware security modules (HSM) offer a higher guarantee of security, since a user can store the sensitive data in a standalone physical computing device. However, it is costly and lacks the scalability of the software-based solution. For instance, if a cryptographic algorithm has a vulnerability to attacks, the software solution can plug in a new cryptographic module with much more relative ease than HSM.
1.2 Approach Overview

In this thesis, we are applying the Trusted Execution Environment (TEE) concept in the cloud system, in order to protect data managed by a KMS. For instance, we introduce the remote attestation concept between the client and server, which provides the certification via the third-parties before the message exchange between the server and the client.

In addition, the remote attestation is utilized between the client and server to protect the sensitive data in transit. Secondly, the study isolates the encryption and decryption processing which execute on the cloud system, according to the isolated mechanism of the TEE. It prevents the data leakage and attack from the inside threats efficiently. In Section 2.1, the report provides a further discussion of TEEs.

1.3 Research Goal and Contribution

The main goal of this research is to study, analyze and apply the state-of-art technology, which can provide strong data protection for the cloud platform when the data is in transit or at rest. Furthermore, the service of the data protection architecture is planned to be deployed in the cluster architecture of Kubernetes. According to that, we follow the concept of the OpenStack SGX-Barbican [5], which applies the TEE technology to the KMS (Key Management Service), in order to protect the sensitive data of the KMS inside the isolated execution environment.

In the contribution of the thesis, firstly, we follow the concept of the SGX-Barbican, and then design the architecture of the Vault-SGX system, where we apply the TEE mechanisms of Intel, such as Enclave and Attestation, to the Vault KMS. Secondly, according to the specification of Vault, we implement the AESNI with GCM, which supports three different lengths, 128, 192 and 256 bits, in pure Assembly code.

Other issues of the cloud-based platform, such as the bottleneck of the streaming data, the migration, and backup of the database are not included in the scope of this thesis.

1.4 Structure of the Thesis

The structure of the thesis is as follows: In Chapter 2, the study explains the relevant background, such as TEE, Kubernetes and the cryptography solution of AES-GCM. Chapter 3 explains the motivation of the adversary action,
and the basic requirements of the system, such as performance and security. Chapter 4 designs the prototype of the TEE run on the KMS. In Chapter 5, the study shows the architecture of the prototype implementation. In Chapter 6, the evaluation provides a comparison between the expectations and reality of the proposed implementations. Chapter 7 discusses the conclusion and the improvement of the future functionality in this study.
Chapter 2

Background

This chapter introduces the basic research background for this thesis. We begin with TEEs, since they are at the core of the thesis. In this context, we list three different security solutions by the Intel, AMD and ARM CPU manufacturers. Second, the study introduces Kubernetes, the cloud-based platform used in our research. Kubernetes is one of the orchestration systems, which has become commonly used in the cloud field. In this chapter, the study also discusses the backend storage and cluster architecture under the Kubernetes platform. Third, this chapter reviews the Key Management Service (KMS) on the cloud-based platform. In particular, the concept of plugging in the TEE with OpenStack SGX-Barbican [5], which has the inspired idea of the thesis. Finally, we introduce the relevant method of cryptography adopted in the security management service of this study.

2.1 Trusted Hardware - Trusted Execution Environment

A Trusted Execution Environment (TEE) is a secure area protected by the main processor (CPU). In general, TEE provides the hardware isolation to prevent the software attacks from the operating system and other untrusted applications. In the study, we utilized the TEE solution to isolate a trusted environment for sensitive data which store and execute on the untrusted devices. Furthermore, the TEE provides the confidentiality and integrity for the sensitive data.

Currently, the CPU manufacturers implement and deploy different solutions of the embedded hardware technologies to supported TEE implementations. Meanwhile, the different TEE solution providers, which introduced their own TEE implementation, and launched it in commercial or open source
licensing. Such as Qualcomm QSEE and Nvidia TLK. According to that, a non-profit organization GlobalPlatform \footnote{https://globalplatform.org/technical-committees/trusted-execution-environment-tee-committee/} has taken the position to publish and verify specifications of the TEE providers. In the next section, the study introduces three TEE platforms by Intel, AMD and ARM, available today.

2.1.1 Intel Software Guard Extensions (Intel SGX)

Intel Software Guard Executions (SGX) is an instruction set extension released by Intel since 2015 on the Skylake microarchitecture. Before the SGX, Intel has implemented other security technologies based on the hardware mechanisms for trusted computing, such as Trusted Platform Modules (TPM) and Trusted Execution Technology (TXT) [7] [2]. However, these security technologies lack flexibility and provide low performance [41]. It makes the challenge for the developers to utilize the technologies in the research field.

In view of the high demand for security in the isolated environment, SGX only consists of two components in the trusted computing base (TCB); CPU and enclave itself [19]. In other words, according to the policy of SGX, except the CPU and enclave, the other components are untrusted in the systems. This is depicted in Figure 2.1. Under normal circumstances, the TCB consists of all hardware, hypervisor, underlying OS and the applications. However, It increases the risk of being attacked. For instance, if malware finds vulnerabilities in the OS, the vulnerabilities can be a tunnel for malware to attack the other applications, for instance, a database or security service. It might cause data leakage of personal information.

In addition to its benefits of secret protection, SGX provides an SDK and APIs for the developers, which aid the developers porting an existing application easily into the Intel SGX enclave. In the next section, the study describes the enclave, attestation, and sealing which are the most important properties of Intel SGX.

2.1.1.1 Enclave

An application is divided into trusted and untrusted component when applying the Intel SGX to protect the trusted component. Developers can create more than one trusted component to run in the same application. In the Intel SGX, the unit of the trusted component can be represented as the enclave.

Enclaves are the trusted execution environment protected by the CPU. Before the system boots up from the kernel, the processor reserves some
space from DRAM as the isolated environment for the enclave, called the Enclave Page Cache (EPC) [13]. The maximum size of the EPC is 128MB. It means that the Intel SGX sets a strict limit on the enclave size. In order to provide a strong security system, the size of the enclave should be as small as possible, which is due to a huge enclave produces a larger attack surface.

In addition to against the attackers, the enclave restricts the type of software to execute in a trusted environment. For instance, when the enclave is in processing, the process can only execute enclave in user mode. As the enclave does not trust the OS or even the kernel, it is prohibited that the process triggers an interruption or executes a system call inside the enclave. The limitation makes it challenging to design the entry and exit point from an untrusted component to the enclave and vice versa.

According to the description of trusted and untrusted components depicted of Figure 2.2, which provides the introduction of the connection bridge between the untrusted component and enclave. According to Figure 2.2, we summarize a few key points. First, the enclaves are initialized by untrusted code. Second, except the untrusted code can execute the trusted code from the call gate, the other external access to the enclave data is denied.

Enclave establishes two measurement registers, MRENCLAVE, and MRSIGNER during the initialization, which is the identification of the enclave. In the activity of attestation and sealing, the two registers play an important role to distinguish a trustworthy enclave.
CHAPTER 2. BACKGROUND

Figure 2.2: Intel Software Guard Extensions application execution flow. [20]

- **MRENCLAVE** - Enclave Identity:
  
  MRENCLAVE is an identity, it records all the activity when the enclave is built. The activity includes, cryptographically, the code and data placed inside the enclave, and the position of the enclave’s pages. MRENCLAVE is unique, the upper variable can cause the different value in MRENCLAVE.

- **MRSIGNER** - Sealing Identity:
  
  MRSIGNER is an identity, which provides the information about sealing authority, a product ID, and a version number. The sealing authority is used to sign the enclave before its distribution. Thus, multiple enclaves can have the same value of MRSIGNER when it signed by the same sealing authority.

### 2.1.1.2 Sealing

When enclave is in processing, the enclave is in protection by the hardware. However, the sensitive data or secret, which executes inside the enclave, are destroyed after the enclave is closed. In some circumstance, the developers need to preserve the sensitive data for the future demand. For instance, when migrating the sensitive data inside the enclave to the external platform, the
CHAPTER 2. BACKGROUND

Sensitive data have to be stored outside the enclave before the enclave is closed.

In order to protect the sensitive data which is stored outside the enclave, the seal key mechanism provides the solution to retrieve the seal key unique to the specific enclave. In the processing enclave, the seal key encrypts and protect the sensitive data when the data is stored outside the enclave.

According to the sealing mechanism, Intel SGX provides two policies to generate the sealing key.

- Sealing the MRENCLAVE:

  The data can be unsealed when an enclave with the same MRENCLAVE measurement, it means that the data can not be retrieved back if MRENCLAVE measurement has changed.

- Sealing the MRSIGNER:

  The data can be unsealed by any enclave who signed by the same sealing authority. It can be utilized when the new enclave downgrade to the previous version.

2.1.1.3 Attestation

In the Intel SGX, attestation can be referred to as a process to demonstrate the two individual enclaves that are both trustworthy. When the confirmation is done, the enclaves establish a session key based on symmetric cryptographic signature algorithm. Since then the data sent between the peers is encrypted and decrypted by the same session key. In the following stage, the study gives a deep introduction to the two mechanisms of attestation.

- Local Attestation:

  Local attestation provides the trust mechanism between two enclaves when the enclaves run on the same platform. This can be done, for instance, when the application has more than one enclave to accomplish the task, or two separate applications run the enclaves on the same platform. Each enclave sends its MRENCLAVE identity to the peer to confirm that they are both trustworthy. Furthermore, the enclave parties ask the hardware to generate a credential report, which contains the data to prove the enclave exists on the platform. The report is sent to its peer to affirm, whether the peer is run on the same platform or not.
• Remote Attestation:
In contrast to local attestation, the two enclaves are run on different platforms. Similarly, the two enclaves generate the trust via a hardware-based mechanism. In remote attestation, the SGX provides the Quoting Enclave (QE), it develops an authenticated channel between a third-party server to verify that the enclave is running on a trustworthy platform.

In order to signing enclave quotes, the remote attestation applies Intel Enhanced Privacy ID (Intel EPID), which is a group signature scheme utilizes asymmetric cryptographic signature algorithms. For instance, a third-party server uses the public key of EPID to verify individual signatures. Therefore, each enclave quote has their own private key for signing.

With the Intel remote attestation, the user is required to issue a service provider ID (SPID) for each device. It is the identity to verify the signature on attestation reports. Therefore, the EPID signature contains the SPID in the attestation reports. While the Intel Attestation Service (IAS) receives the challenge request from an external party, the SPID is used by the external party to verify the trustworthy of its peer via IAS.

In particular, before the secure channel have been created, the remote attestation utilize the Sigma protocol between the two enclaves to perform a Diffie-Hellman Key Exchange (DHKE). Thus, the session key of the secure tunnel is created through the procedure of DHKE. In the next section, it applies the use case to introduces the above properties of Intel SGX plugin system.

2.1.1.4 The Lifecycle of Intel SGX Enclave
In the scenario of Intel SGX plugin system, the client request the sensitive data from the service, the data is remotely provisioned from the enclave execute in the server to the enclave in the client. As the data will be utilized in the future, the enclave encrypts the data in the storage. In the illustration of Figure 2.3, the enclave lifecycle is depicted in detail.

1. Enclave Launch - The untrusted application launches the trusted environment to generate the enclave in order to protect the sensitive data. During the enclave initiation stage, the software records the reaction log and context of the enclave and then to generate the measurement,
it is \texttt{MRENCLAVE} and \texttt{MRSIGNER} that the study introduces in the previous section.

2. Attestation - the client enclave contacts the remote enclave, which executes on the service provider, to establish the secure channel in order to do the data provisioning from the service provider to the client. During the stage to create the secure channel, the client enclave sends the identity of its hardware platform to identify itself to the remote enclave.

3. Verification - After the service provider receives the identity from the client enclave, it passes the information to the Intel attestation service to verify the trustworthiness of the client.

4. Provisioning - After the IAS has verified the trustworthiness of the hardware platform which executes the client application, the secure channel is ready to provision the sensitive data from the service provider to client enclave.

5. Sealing/Unsealing - As the sensitive data vanishes when enclave finishes its process, the enclave needs to outsource the storage of the sensitive data for future use. During the sealing stage, the sensitive data is encrypted by the seal key, which generated by the enclave uniquely. The data can be retrieved back using the same seal key.

\subsection{ARM Trustzone}

TrustZone is a set of security extensions that was introduced in the ARM architecture. Due to the benefit of low power processing, ARM processors
are commonly used in IoT devices. That is the reason why the TrustZone technology is widely utilized in the smartphones and tablet computers.

Similarly to Intel SGX, in order to achieve the confidentiality and integrity of the system, TrustZone establishes an isolated environment for sensitive resources. In this concept, TrustZone divides the software and hardware resources of SoC into a secure world and normal world, which can be referred as to the Secure OS (secure world) and Rich OS (normal world). The Secure OS is the environment to execute the secure resources and the Rich OS run the rest of resources. In addition, the secure and the normal world needs an extra tunnel for switching between each other, which is monitor mode. It records the world statement before the OS switches to another party.

The key advantage of ARM TrustZone is that two OSs can run on the single processor at the same time, as each physical processor provides two virtual processors. Such as a 'non-secure' and a 'secure' processor. The secure processor can access all the resources on the system. However, the nonsecure processor forbids access to the sensitive resources.

### 2.1.2.1 OP-TEE

OP-TEE\(^2\), which uses the ARM TrustZone technology, is an open source TEE solution. It supports plenty of platforms which from different vendors implementing ARM architecture. In Figure 2.4, the OP-TEE consist of three component for the vendor to porting the Trustzone technology on the different platform.

- **OP-TEE Client**: The API, which runs in the rich os with userspace.
- **OP-TEE Linux Kernel Driver**: The driver, which build the communication tunnel between the rich os user space and trusted os kernel space.
- **OP-TEE Trusted OS**: The OS run in the secure world, which consists of two main components: OP-TEE core and static libraries. The static libraries provide the solution for Trusted Applications to execute the functionality from the user space to the kernel spaces, which is executed by OP-TEE core.

### 2.1.3 AMD SEV

AMD SEV is a hardware security feature based on the virtual machine (VM) concept. In the cloud computing industry, a hypervisor can run more than

\(^2\text{github: https://github.com/OP-TEE} \)
one VM at the same time. Furthermore, the hypervisor provides a secure execution environment for each VM, as shown in Figure 2.5. However, the hypervisor does not block itself to access the resource in VM. For this reason, the AMD introduces the SEV technology, which provides the unique key for each VM to encrypt memory with the AES algorithm. Furthermore, the security processor manages all the encrypted key.

Compared with ARM TrustZone and Intel SGX, the AMD SEV lacks integrity protection in VMs [10]. For instance, the hypervisor can access and modified the memory data of VMs, even though the data has been encrypted. In early 2018, the Fraunhofer Institute utilized the weak integrity in AMD SEV to recover all the encrypted data [30].

2.2 Kubernetes

Containerization is a useful technology for shipping and packaging applications. It improves not only the utility of resource but also eases the difficulty to migrate the application. [3]

As an orchestration system for containers, Kubernetes provides the ben-
benefits of the high availability, load balancing and private networking for containerized applications in the cluster architecture of Kubernetes.

2.2.1 Cluster Architecture

A basic unit of the cluster is a node. Each node can be considered as an individual device. The principle of clustering adopts the concept of a distributed system, where the cluster is considered as one system instead of multiple systems. The administrator of the cluster can add and delete nodes in the system. Furthermore, the nodes included in the cluster can contact each other via a private network. The advantage of the clustering is high availability, load balancing, and high scalability, as the cluster can migrate the workloads from one overloaded or failed node to another available node in the cluster.

In the Kubernetes architecture, a high-availability cluster requires at least three master nodes and three worker nodes in a production environment [26]. The master manages containerized applications in the worker nodes.

- **Kubectl**: The command line tool, which is the way for the user to interact with the master node(s).
- **Pod**: The set of containers, which can share a network namespace and file system.
- **Etcd**: A backend store for the Kubernetes cluster data.
2.2.2 Etcd Storage Backend

Etcd is a distributed key-value store introduced by CoreOS. It provides high availability and reliability as an Etcd cluster. Furthermore, the Etcd cluster uses the Raft algorithm to ensure the consistency of the stored data.

Etcd stores the data in the wal log, which is created in binary format. (However, the user can convert the binary format to plain text via the tools/etcd-dump-logs tool.) In the consistency model of the Etcd cluster, the master node updates the wal logs, which are stored in different work nodes. In addition, Etcd utilizes the Raft Algorithm to update the wal log synchronously.

In the this thesis, the key management service (KMS) consists of Etcd and Vault. In the allocation of the KMS, The Etcd is utilized as the backend storage, and the Vault is considered as the secret management service to protect the data in storage. In the next section, the study introduces the Barbic, which is the KMS of OpenStack enhanced with Intel SGX based protection. Furthermore, the Vault is described in the next section.

2.3 Key Management Service

Key Management Service (KMS) refers to a service used for managing the cryptographic keys or other secrets, such as the pin code of the bank account or the email password. The obvious way to keep a secret is to remember it. However, it is an obvious challenge if the user has several secrets, long secrets, or only machine-readable secrets. Another way to store a cryptographic key is to outsource the secret to a database. As the data is important enough to be encrypted by the cryptographic key, the cryptographic key itself is also considered a secret.

In order to protect the cryptographic key in the KMS, the KMS includes the secret management service to encrypt the secret in the system. In the next section, the study introduces two well known secret management services, namely Barbican and Vault.

2.3.1 OpenStack Barbican

Barbican is a centralized key management system for OpenStack. It provides the secure storage, provisioning, and management of secrets via a REST APIs. In order to provide solid cryptographic in a diverse environment. It

3 github: https://github.com/coreos/etcd
4 Barbican’s developer documentation: https://docs.openstack.org/barbican/latest/
supports plugins for a variety of cryptographic methods, including new crypto solutions.

As a KMS of the OpenStack cloud OS, Barbican provides the high availability and load balancing benefits in the system. As shown in Figure 2.6, the API nodes interact with Barbican via a REST APIs. Moreover, the API nodes can interact with the database directly without going through the other components. Otherwise, the messages queue in worker nodes and progress asynchronously, after that the worker nodes interact with third parties on demand. In addition, Barbican supports the auto scaling feature of the API and work nodes, which means that nodes can be added or removed on demand.

![Figure 2.6: Barbican Architecture. [31]](image)

### 2.3.2 Intel SGX Based Crypto Plugin of Barbican - BarbiE

According to the research of BarbiE [5], the authors describe how Barbican is vulnerable to the insider attacks. Although Barbican supports the Hardware Security Module (HSM), which provides protection against the insider attacks, the high price and the low scalability cause low acceptance of it in
the market. Thus, the concept of BarbiiE, which uses Intel SGX to replace the HSM, has come up.

![Figure 2.7: Intel SGX Enabled Barbican Architecture. [5]](image)

In the high-level architecture shown in Figure 2.7, the SGX enabled Barbican (BarbiiE) generates enclaves in client and server, and builds a secure channel between them, applying the remote attestation mechanism of SGX. The secure channel utilizes the DHKE to negotiate the session key between the two parties, which encrypts the sensitive data in transit. In addition to protecting the data in flight, BarbiiE also considers protection of the data in use. In general, the data stored in the database is encrypted by a Master key, as the raw data without encryption can be tampered with easily. The BarbiiE design executes the master key processing, including encryption, decryption, and initiation, inside the enclave.

As shown in Figure 2.8, the client and service negotiate the session key (SK) via the remote attestation process. In the scenario, the client plans to store the sensitive data in the remote database. As the SK provisions the data protection in flight, the sensitive data is encrypted by the SK when it passes through the secure channel. After that, the server applies the master key to encrypt the sensitive data before it is stored in the database. The Intel SGX-based KMS support the same level of security protection as an
HSM plugin.

![Diagram of Secret Management in BarbiE](image)

*Figure 2.8: The Secret Management flow in BarbiE. [5]*

In the research of this thesis, we port the concept of data protection from BarbiE to Vault, including the remote attestation between the client and server, and provisioning the master key into the enclave. The full prototype is introduced in Chapter 4.

### 2.3.3 Vault

Vault is an open source of secret management service\(^5\), which can be ported to any system handily (for instance Windows, Linux, and macOS). As a security management solution, it has similar functionality as Barbican. For instance, it supports the pluggable backend architecture, which enables using different storage backend or authentication methods.

Next, we depict three important stages of Vault, which provide the nesting of the encryption, in order to protect the sensitive data.

- Initializing the Vault:

\(^5\)github: https://github.com/hashicorp/vault
In the beginning, Vault requests the client to execute an initialization command. During this stage, Vault generates the Master Key and the Encryption Key, which is the preparation for the data encryption. Therefore, the server also generates the root token in the initialize stage for the user, which is the identity of the user to prove the authority of the execution commands.

![Seq Diagram](image)

Figure 2.9: Vault Initialization - Sequence of Operations.

According to the Shamir’s Secret Sharing algorithm [35], Vault utilizes it to split the master key in the shared key list. This method ensures no single person has the ability to retrieve the master key back alone. The configuration of Shamir’s algorithm includes the split number in
shared key and the threshold number of the key to regenerate the master key back, the configuration data indicates data0, as illustrated in Figure 2.9. In addition, Vault utilizes AES-GCM to encrypt the Master Key with the Encryption Key and to store it in the backend database afterward, and vice versa.

- Unsealing the Vault:

![Diagram showing the unsealing process]

Figure 2.10: Vault Unsealing - Sequence of Operations.

Vault blocks the manipulation before client unseals the vault. In the unsealing mechanism, the user is required to provide the shared keys in order to retrieve the Master Key back. According to the Shamir algorithm, it follows the configuration to split the Master Key in fixed pieces, and the threshold number of shared keys to unseal the Vault, as shown in Figure 2.10. The shared keys regenerate the Master Key
back, and it decrypts the encrypted Keyring Key which consists of the Master Key and Encryption Key. In particular, the Keyring Key and Master Key are all exposed in the memory without protection.

- Encrypted/Decrypted data in backend via Vault:
  
  Vault server requires the clients to identify themselves before storing or requesting data from the server. According to that, the client sends the root token to log in to the server. Afterward, the server fetches the naming policy via the received token. It is the access control list (ACL) regulation in the Vault server.

  According to the illustration for unsealing the vault, the data required to encrypt and decrypt before to store and obtain from the database, respectively. During the encryption stage, Vault utilizes AES-GCM to protect the sensitive data, which is the outsourcing storage by the client.

Comparing Vault and OpenStack Barbican, we note two differences. First, OpenStack Barbican is a distributed system providing the scalable worker node to handle the request horizontally. In contrast to OpenStack Barbican, Vault does not support the horizontal execution. However, it supports the high-availability functionality of the backend storage. Furthermore, Kubernetes can fill the gap to deploy Vault as a cluster and handle the request in different nodes. Second, in terms of the protection method of the master key, the OpenStack Barbican writes the master key in the configuration file without any protection. Compare to that, Vault utilizes the Shamir secret sharing algorithm to split the master key into a number of shares. The master key can be retrieved and decrypted by combining the shares. The concept tackles the issue when a single person tries to regenerate the master key back. However, the master key is held in the memory when it is retrieved by the shares. At this moment, the hacker can monitor the master key via the memory. Neither solution offers high protection of the master key.

In the OpenStack Barbican, the HSM plugin addresses the security issue in the system. Although Vault also supports the HSM plugin, it only available for the Enterprise version. In our research, we plan to utilize the Intel SGX crypto, the software-based crypto plugin to replace the security protection, which provides by the HSM.
2.4 Cryptography

2.4.1 AES-GCM

Both OpenStack Barbican and Vault utilize AES-GCM to encrypt the data before being stored. In this section, the study introduces the AES-GCM by example, in order to make the crypto method clear to understand.

2.4.1.1 Advanced Encryption Standard - AES

Advanced Encryption Standard (AES), which is based on the Rijndael cipher algorithm, is classified as symmetric cryptography algorithm, as the encrypting and decrypting use the same key. Furthermore, in AES, the block length of the plaintext is fixed to 128 bits, and the key can be selected in three different lengths, 128, 192 and 256 bits. Depending on the different length of the key, the plaintext requires 10, 12 and 14 rounds of iteration, respectively, to generate the ciphertext. In the default setting of Vault, it applies the 256 bits key length to encrypt the plaintext.

In Figure 2.11, the study uses a 128 bits key as an example to introduce the encryption procedure from plaintext to ciphertext via the AES algorithm.

1. The 128 bits key is converted into a 4x4 matrix. After that, the Rijndael’s key expansion method generates a set of key, which is based on the number of iterations. For instance, if the procedure iterates 11 times (from round 0 to round 10), the key expansion algorithm also produces 11 different keys.

2. The 128 bits plaintext is converted in a 4x4 matrix.

3. From round 0 to round 10, the plaintext matrix is executed with four different transformations (Figure 2.12), which are SubBytes, ShiftRows, MixColumns, and AddRoundkey.

- **SubBytes:** The element (a2.2) of origin matrix (A) is substituted to a matrix (B) by another value via a lookup table called S-box, the S-box is a designed box in 16x16 matrix.
- **ShiftRows:** The ShiftRows is a linear transformation, which achieves the diffusion of the bytes in each row.
- **MixColumns:** The element (a2.2) of origin matrix (A) is substituted for a matrix (B) by multiplication with a fixed matrix.
- AddRoundkey: The matrix executes bitwise XOR with subkey matrix, which is generated by Rijndael’s key expansion method.

The decryption procedure executes the reverse way compared to the encryption procedure in the above introduction.

2.4.1.2 **Counter Mode - CTR**

GCM is a mode of operation composed of counter mode (CTR) and Galois message authentication code mode (GMAC), which provides the integrity and confidentiality of the ciphertext. Next, the study introduces CTR and GMAC separately, which gives a more clear view of the GCM.

In the AES cipher, the plaintext is converted to ciphertext via an encryption procedure. However, the AES cipher only accepts the length of plaintext in 128 bits. According to that, in the example of Figure 2.13, the plaintext is divided into pt1 and pt2, as it has more than 128 bits.

In the depiction of Figure 2.13, AES encrypts the counter block, which generated by the random IV (Initialization Vector) or nonce (the default
length in Vault is 96-bits), and then to combine with the increment-by-one counter using any lossless operation, such as concatenation, addition, or XOR.

The above procedure produces a unique counter block which prevents the attackers to eliminate the encryption via the same ciphertext. However, the ciphertext is still in risk to be tampered with without discovery. Next, the study introduces how to use GMAC to protect the integrity of the plaintext.

2.4.1.3 Galois Message Authentication Code Mode - GMAC

In the discussion above, the GCM utilized the CTR to provide the encryption of the plaintext. In this section, the study describes how to generate the MAC (Message Authentication Code) in the GMAC algorithm. In order to avoid tampering, in the concept of GMAC, the sent message also include the MAC value to protect the integrity of the data.

The authentication mechanism of the GMAC is based on a hash function and the multiplication of the Galois Field (GF). In the GMAC, the cipher key is called H, where H is the encryption with a key, and the input of plaintext
is 0 in 128 bits, in the following equation:

In Figure 2.14, \( M_H \) is the input value, which multiple with \( H \) in a Galois Field (Finite Field), both of the 128 bits binary digit converts to a polynomial. The following shows the conversion from 4 bits binary digit to the polynomial:

\[
(1011)_2 = x^3 + x + 1
\]  

(2.1)

2.4.1.4 Galois Counter Mode - GCM

The CTR and GMAC functions provide the encryption and authentication of the plaintext, respectively. However, it is not enough for the receiver to trust the receiving message. In order to achieve the standardize of AEAD (Authenticated Encryption with Additional Data), as in the depiction Fig-
Figure 2.15: GCM Flowchart.

2.4.2 Intel Advanced Encryption Standard New Instructions - AESNI

Although the AES-GCM provides confidentiality and integrity protection, the sensitive data are still susceptible to cache-timing attack [15], and intensive calculation. In order to address the drawbacks of AES-GCM, Intel has created a set of 7 new instructions called AES-NI [27] which can replace the software-based AES-GCM with hardware instructions. It does not only reduce the code size and number of CPU cycles in each round of processing but also decreases the vulnerability to cache attacks [15].

According to the evaluation result by Intel [27], in the algorithm level, the AES-NI gives four times faster execution than the traditional AES-GCM in serial mode. Furthermore, in parallel mode, AES-NI boosts the performance up to 10 times in the encryption rate.

In this research, the study adopts the AES-NI to replace the AES for encryption usage in KMS.
Chapter 3

Adversary Model and Requirements

In order to establish a secure system, ideally, we need to consider the variety of circumstances in which the system may be attacked by an adversary. Analyzing and identifying the factors which can cause vulnerabilities in the system aids in gaining a clear understanding of the system, and in designing the relevant protection to anticipate the attacks in order to improve the security.

In this chapter, the thesis introduces the essential assets that interest the adversary to challenge the threat model in the system. We also list the capabilities of the adversary which may be used to endanger the system. In order to address the corresponding attacks, the relevant defense capabilities are required in the system to provide needed security guarantees.

3.1 Adversary Model

Depending on the different capabilities and objectives of an adversary, attackers, and attacks can be placed into several different categories such as multinational criminal organizations, hacktivists/terrorists, and insider attacks. Even though each adversary has different motivations to break the system, the aim of the adversary is basically the same: to defeat the goal of the cryptosystem, which is built to protect the system and the sensitive data.

In the next subsections, the thesis introduces the adversary model, according to which attackers intend to break the system and steal sensitive data from the database.
CHAPTER 3. ADVERSARY MODEL AND REQUIREMENTS

3.1.1 Goals and Motivations

According to the context of this thesis, the client outsources the storage of secrets to a remote system. When the data is no longer possessed only locally, a variety of risks may endanger the confidentiality and integrity of the data, the data is in transit, in use\textsuperscript{1} or at rest. In addition, a bigger target, such as a storage provider or gaming server, also raises the potential profit of an attack.

In the case of a storage system, for example, the main goal of the adversary is to steal the valuable data in the system, or possibly to crash the system itself or other systems.\textsuperscript{2} Monetary gain and curiousness towards private data can motivate an adversary to attack and hack the system.

3.1.2 Capabilities

The capabilities of the adversary are the starting point for designing an appropriate security model for the system. In our research, we assume the adversary has the right to control and access all the components in the system, except the TCB (Trusted Computing Base), which consists of the CPU and the enclave, which is a software component created by Intel SGX. In addition to its capabilities [37], the adversary is assumed to have knowledge of its attack target regarding, for instance, what type of cryptography has been applied in the system.

The capabilities and knowledge of the adversary are listed below:

- Access and control: The adversary has the full right to access and control all the components in the system except the CPU and enclave.

- Knowledge of the system structure: The adversary knows the whole structure of hardware and software which runs in the system.

- Monitor the data in transit: The adversary has the ability to monitor the messages between communicating entities.

- Privilege to manipulate the system: The adversary has the complete privilege to compromise the system without not.\textsuperscript{ce.}

\textsuperscript{1}The data in use means the data store in the non-persistent storage, such as RAM or CPU cache with active mode

\textsuperscript{2}For instance, in 2014, the hacker who hacked iCloud leaked the private data of celebrities in the public, and the leaked data was exchanged for Bitcoin on the Internet. \url{https://www.businessinsider.com/originalguy-the-icloud-hacker-who-leaked-naked-celebrity-photos-2014-9?r=US&IR=T&IR=T}
3.1.3 Related Attacks - Security Vulnerability

In this section, we introduce the Foreshadow and Foreshadow-NG vulnerabilities, which utilize a well-known timing attack affecting the Intel processors. Due to these vulnerabilities, the encrypted memory protected by SGX can be tampered with without discovery.

Foreshadow and Foreshadow-NG [4] are speculative execution side channel attacks affecting Intel processors caused by the L1 Terminal Fault [4]. Specifically targeting Intel SGX, the attack can extract the sealing key of an enclave, and then unseal the sensitive data protected via Intel SGX. Moreover, the attackers can recalculate the Message Authentication Code (MAC), so the owner of the enclave is unable to detect any modification in it, even though the attacker unseals the enclave, modifies it, and then seals the enclave back. In this thesis, we are not considering the attacks from Foreshadow and the Foreshadow-NG, as Intel has given an official announcement [17] that they have updated the processor microcode in their new patches to prevent those attacks in Intel processors.

3.1.4 Attack Scenario

In order to consider the inside and outside attacks against the system, we include two different types of attacks in our discussion, namely passive and active attacks [36]. In general terms, the passive attacks only observe the data without compromising the confidentiality of the data. Active attacks, on the other hand, can be considered an even more dangerous attack type than the passive attacks, as the active attacks tamper the data to compromises the integrity of the system.

In the following section, we describe a scenario where the adversary performs these attacks in different circumstances. The capabilities of the adversary, described in the previous subsection, are also considered. In the scenario, we assign the role of the adversary to a whose name is Mallory, and the attack target is the server of the Examination Center.

- Inside - Passive Attacks: The adversary monitors the activity in the whole system. After the long period of data monitoring, the adversary observes the data pattern and reads the content of data. This gives the adversary a chance to obtain sensitive data from the system. In our example scenario, Mallory monitors the server of the Examination Center remotely, in order to steal the exam answers before the examination.

- Inside - Active Attacks: In the same scenario, after the examination, Mallory got a good grade. However, he forgot to share the answer sheet...
to his friend. His friend coerces Mallory to help her modify the grade which is recorded in the examination center server (or she will report the teacher what Mallory has done).

Since Mallory has the privilege to compromise the Examination Center Server, he monitors the memory of the procedure to record the grade on the server. After this analysis, he mimics the stored procedure to replace the grade with the higher value.

- **Outside - Passive Attacks:** In the similar circumstance as in the Inside - Passive attacks, Mallory has another exam, but this time, he knows the teacher has not uploaded the answer to the Examination Center Server. He monitors the network activity between the teacher’s computer and the server and obtains the exam answer when the teacher uploads it.

- **Outside - Active Attacks:** In the similar circumstance as in the Inside - Active attack, Mallory wants to modify the exam grade, but this time the teacher has not uploaded the student’s grade to the Examination Center Server. Mallory monitors the network activity between the teacher’s computer and the Server and captures the package containing the grade sent by the teacher. He replaces the grade with the higher score, regenerates the package, and forwards the modified package back to the Examination Center Server.

### 3.2 Performance Requirement

In the storage system, request throughput is a key performance metric that we consider.

Existing research suggests that [32], comparing execution with and without enclaves, the relative overhead increases when the system enters and exits the enclave and untrusted component constantly.

An efficient way to increase the throughput in the Intel SGX-based system is thus to reduce the switching frequency between the trusted and untrusted component. In the research of SGX-Barbican [5], compared with the normal Barbican, SGX-Barbican added a 10% overhead to the mean processing time. In our research, Vault-SGX should incur an equivalent overhead compared to the normal Vault. The main factors to measure regarding performance are listed below.

- **Latency:** The delay of the system response. In the perspective of our research, we consider it as the processing time when the client sends and receives a message from the server.
• Throughput: The amount of data processed per unit time between the devices.

• Scalability: The system needs to avoid bottlenecks when it receives multiple requests at the same time.

### 3.3 Security Requirement

The security requirements that we define in this section set guidelines for the design decisions in our research prototype. We specify different requirements regarding system protection, which we apply to different components of our system. In our research, we assume the adversary has the privileges to access and monitor all the component with the inside and outside attacks of the system. Our goal is to protect the sensitive data from being stolen and extracted by the adversary. According to the related research of Intel SGX [25] [1], we list the corresponding security requirements as below.

• Prevent offline guessing:

  In an offline attack, the adversary manages to guess the sensitive data without the interaction with the system or server. The attack can be prevented by using the one-way cryptographic function or applying the extra key to encrypt the cryptographic key, which applies to encrypt the sensitive data.

• Guarantee the trustworthiness of the remote party:

  In order to achieve this requirement, we need a third party to confirm that the devices are trustable before they exchange the data, such as the Intel Attestation Server (IAS).

• Guarantee the confidentiality and integrity of the data:

  In order to keep the fundamental concept of security, we design the Vault-SGX with the consideration of confidentiality and integrity. In the research, we apply the AES-NI with GCM and enclave mechanism of Intel SGX to achieve this requirement.

• Minimize the attack surface:

  In order to optimize the performance and the security, the TCB of an enclave is considered to consist of the SGX hardware and the code inside the enclave. Therefore, minimizing the TCB size is one of the factors to consider when designing the system [28].
Moreover, according to the Intel SGX developer guide [16], a smaller memory footprint can reduce the attack surface and the risk of having vulnerabilities.

3.4 Deployability Requirement

In our research, we plan to execute our Vault-SGX prototype in Kubernetes clusters. In this way, Vault-SGX can utilize the load balancing and failover features of the cluster architecture. Furthermore, we also assume that the database is scale horizontally and also deployed in the cluster architecture.

3.5 Confidentiality, Integrity, and Availability (CIA) Triad

CIA Triad is a model to design the policy of information security. In order to design a concrete security concept in the prototype, we utilize this fundamental concept in our prototype. The security requirements above cover the confidentiality and integrity principles, which are two elements of the Triad, with respect to the data protection. In addition, the availability principle of the CIA Triad is partly addressed in the deployability requirement, as it supports the user to maintain access to the data.
Chapter 4

Design

In this chapter, we describe the general design concept and architecture of the Vault-SGX system. We present its components and subsystems, and provide a detailed introduction to the functionality of the system. In the system design, we consider the requirements and the attacks circumstances of the adversary model from the previous chapter.

4.1 Design Overview

The design concept of the Vault-SGX system refers to the BarbiE [5] (a.k.a. SGX-Barbican) architecture, and in the same way as BarbiE it utilizes the Intel SGX technology to provide strong security guarantees similar to a HSM while having the low cost and scalability benefits of a software-based solution. Similarly, in the Vault-SGX prototype, we aim to execute the entire progress of the sensitive data in the system inside isolated execution environments. To achieve it, we keep the sensitive data in the enclave, when it is used in the application or is in flight. According to that principle, this research introduces the Vault-SGX prototype architecture shown in Figure 4.1.

In the depiction in Figure 4.1, the architecture comprises three individual devices, which are the client, secrets management system, and database. The sensitive data is executed in and transmitted between the devices using the enclave and attestation mechanisms of Intel SGX. The sensitive data is thus protected by the hardware-assisted trusted execution environment to prevent direct attacks from the adversaries.

In the Vault-SGX architecture, we divide the system into 4 major functions and components, namely local/remote attestation, the enclave id registry server, encryption key retrieval, and encryption/decryption in the enclave. Each of the components includes multiple submodules. In the next
section, we introduce the components in detail and then propose a scenario in order to support the concept of the Vault-SGX architecture.

Figure 4.1: The architecture of Vault-SGX.

4.2 Components

In our design, we consider each component as an individual instance, which can be executed and built without any dependency.

4.2.1 Local/Remote Attestation

Attestation is the process that enables the system to prove that an enclave is established in a trustworthy environment. As described in the introduction in Section 2.1.1.3, Intel SGX defines two mechanisms, local attestation and remote attestation, between a pair of enclaves within the same or different platform respectively. Similarly, both mechanisms use the Diffie-Hellman (DH) key exchange mechanism to establish a protected channel between the enclaves.

In this component, our implementation follows the Intel SGX SampleCode \(^1\). In our research, we port the attestation implementation from Sam-

\(^1\)The SampleCode is provided by Intel at https://github.com/intel/linux-
pleCode to our system, in order to support the transmission of sensitive data between the enclaves.

- Local Attestation:

  In Vault-SGX, we apply the local attestation mechanism in the secrets management system to execute data sharing, instead of utilizing the Sealing mechanism of Intel SGX. (Sealing saves the data to the hard disk, but disk corruption or malicious modifications can cause data to be lost.)

  The local attestation procedure makes a bridge to connect two components which can then exchange the sensitive data via a protected channel. Usually, two individual enclaves utilize the local attestation or Sealing, which depends on the enclaves executing on different or standalone applications within the same node respectively, to share the data. For instance, in the illustration in Figure 4.2, enclave A requests the Encryption Key from enclave B for data encryption. The raw Encryption Key is sent from enclave B to enclave A via the secret channel, set up by the local attestation. In the design of Vault-SGX, in order to avoid exposing the raw (unencrypted) data outside the Intel SGX protection, we always process the raw data inside enclaves, and transmit it between the enclaves over the secure channels.

- Remote Attestation:

  As depicted in Figure 2.3 (Lifecycle of SGX Enclave), the client transmits the sensitive data to the Service Provider, the Service Provider uses the remote attestation mechanism to challenge the trustworthiness of the client. After that, if the client has no vulnerabilities discovered, the Service Provider receives the trustworthiness certification of its counterpart from the IAS (Intel Attestation Server).

  In our design, we utilize the mutual remote attestation, i.e., two-way remote attestation. For instance, the client verifies the trust in secrets management system via the IAS, and vice versa. In the same way, the remote attestation procedure is utilized in the communication between the secrets management system and database. During the remote attestation procedure the devices share a session key and thus establish a secure channel for the sensitive data transmission.

sgx/tree/master/SampleCode. (In Linux, after installing the Intel SGX SDK package, the source code is installed under the directory /opt/intel/sgxsdk/SampleCode.)
4.2.2 Retrieve Encryption Key

In the original concept of the Vault system, it requests the user to execute the initialization process of Vault (see Section 2.3.3), if the user is in its first time to contact with Vault system. During the initialization process, the Vault creates three key, which are Encryption Key, Master Key and Keyring Key (In the depiction of Figure 4.3, The value of an element ”Value” is the Encryption Key). Keyring Key consist in Master Key, Encryption Key and some generated information of the key. After that, the Keyring Key is encrypted by the Master Key and then stored inside the database for future use.

As we know, the sensitive data is encrypted by an Encryption Key before it is stored inside the database. In order to retrieve the Encryption Key back, the Vault system first requests its client to provide the shared key list. The list is the divided parts of the Master Key, which are generated by the Shamir Algorithm during the initialization stage of Vault. After the system retrieves the Master Key back, it utilizes the Master Key to decrypt the
cipher Keyring Key and then to retrieve the Encryption Key back.

In the Vault system, the procedure to retrieve the Encryption Key back is also known as the Vault unsealing step (see Section 2.3.3). As shown in Figure 4.4, it is the similar process of the Vault Unsealing in Figure 2.10. However, in Vault-SGX, we introduce the process of Vault Unsealing inside an enclave.

1. The Client sends the shared key list one by one via the secure channel, which is established by the remote attestation process. After the Secrets Management System receives the shared key, it performs local attestation and then sends the shared key to enclave A. The enclave keeps the shared key for the preparation of the Master Key retrieval.

2. Since the number of the shared key pieces has reached the threshold of the Shamir configuration, the secrets management system utilizes the Shamir algorithm to retrieve the Master Key back inside the enclave A.

3. After the system succeeds to retrieve the Master Key back, the secrets management system requests the database to reply the cipher Keyring Key via the Secure Channel, which is established by the remote attestation process.

4. When the enclave A receives the cipher Keyring Key via the local attestation process, it utilizes the Master Key to decrypt the cipher Keyring Key. The secrets management system executes the entire procedure inside the enclave A.

5. As depicted in the sequence diagram (Figure 2.9) of Vault initialization, the Keyring Key comprised of the Master Key and Encryption Key. In this step, after the system decrypts the cipher Keyring Key, it utilizes local attestation to pass the raw Keyring Key into the enclave B and then to extract the Encryption Key from the Keyring Key. Meanwhile, the system kills the enclave A in order to save more space of the EPC
(Enclave Page Cache) memory. During the entire procedure, the raw Keyring Key is kept inside the enclave B for the future demand.

![Diagram](image)

Figure 4.4: The high level overview of Vault Unsealing process in the Vault-SGX system.

### 4.2.3 Enclave ID Registry Server

The enclave id registry server can be considered as an enclave management system. The concept is to record the status of the enclave and its capability, such as status:1, cap: storage. In addition, as the session developer has to provide the source and destination id of the enclave when establishing the session between the enclave, the enclave id registry server provides the ID table lookup with the capability and the status of the enclave. The service thus supports the user to obtain and update the enclave status without hardcoding it inside the source code.

In our design, the Enclave ID Registry Server is created when the Vault Server is started, or another enclave requests it when the Enclave ID Registry Server does not exist.

The Enclave ID Registry Server is established as an enclave with the process id of the Vault Server. It is pending in the system all the time to
CHAPTER 4. DESIGN

wait for a new registration and request by the enclave, which runs in the same process.

We introduce two different circumstances of the Enclave ID Registry Server. In the first step, the Enclave ID Registry Server receives the event id with different requirements. Such as, REGISTER_REQUEST and ENCRYPTIONKEY_REQUEST. Following, we list each circumstance, according to the related event id.

- REGISTER_REQUEST:
  In the system, when the enclave is created, it registers itself to the Enclave ID Registry Server. The server records the id of the enclave and the capability of the enclave. For instance, if the enclave is the storage to store the Encryption Key, the server keeps the information inside the enclave as a lookup table of the capability id with the enclave id.

- ENCRYPTIONKEY_REQUEST:
  To continue the example above, if the other enclave needs the Encryption Key, but does not know the enclave id of the Encryption Key storage, it requests the Enclave ID Registry Server with the request event id. When the server receives the request, it checks its table to map the requirement and then replies the enclave id back to the requesting enclave. If the Enclave ID Registry Server can not find the mapping id, it triggers the related module to establish the required enclave and registers it to the Enclave ID Registry Server for the future utility.

4.2.4 Encryption/Decryption in the Enclave

The original design of Vault utilizes a 256-bit Advanced Encryption Standard (AES) cipher in the Galois Counter Mode (GCM) with 96-bit nonces to protect data in the backend storage. In the Vault-SGX, we follow the same conception. The main difference, however, is that we execute the AES-GCM with 256-bit key length inside an enclave.

In our research, Vault supports the AES-NI cryptographic operations, which are the hardware AES instructions for encryption and decryption. As we have mentioned in Section 2.4.2, software-based AES is not optimized for performance, and is vulnerable against cache-timing attacks. AES-NI, however, reduces the latency by the average number of CPU cycle decrease, and mitigates the cache-timing attacks.
Even though Intel has implemented the AES-NI for the GCM with 128-bit key length for SGX, as Vault utilizes the 256-bit key length, we prefer to keep this specification, i.e., to utilize 256-bit AES-NI instead the 128-bit option.

As we consider this component as a major contribution in this thesis, we establish it as an individual library, which can be ported to any Intel SGX-enabled platform, in order to provide benefits to users who would like to utilize the AES-NI with 256 bits to encrypt data and also protect it by the enclave.

4.3 Scenario

In the following scenario, we assume that each component introduced above is ready to be applied in the use case. For instance, the raw Keyring Key has been decrypted stored inside the enclave. If, however, the raw Keyring Key does not exist inside the enclave, the system has to request the client to execute the Vault unsealing step again.

In this section, we assume the process of local and remote attestation has been executed and the secure channel is ready to exchange the data between the enclaves. Therefore, the step of the enclave registration and requesting the id from the Enclave ID Registry Server is skipped in the rest of the scenario description.

4.3.1 Client Stores Data to Server

In this use case, the client plans to store sensitive data to the server remotely. We assume that the client and server devices all support Intel SGX. Corresponding to the illustration in Figure 4.5, we describe the remote data storage procedure step by step.

1. The client plans to store its password in the server. The payload in plaintext is structured like \{key: 'password', value: '1234567'}\}. As the data is sent from enclave A to enclave B via the secure channel, the payload of the data is encrypted by the pre-shared key that the two enclaves (A and B) generated via the remote attestation process. The encrypted payload is as the following format, SK.Cenc({key: 'password', value: '1234567'}).

2. When the enclave B receives the data, it decrypts the payload using the pre-shared key. After that, enclave B utilizes the secure channel created by the local attestation process to send the sensitive data to enclave C.
3. After enclave C receives the sensitive data from enclave B, it requests the Keyring Key from enclave D. Enclave C uses the Encryption Key, which is extracted from the Keyring Key to encrypt the value of the sensitive data with the AES-NI GCM cipher, e.g., \{key: 'password', value: AESenc('1234567')\}.

4. In our conceptual design, the database is run on a backend device individually. The secrets management system is required to create a secret channel between the devices for transmitting the sensitive data. According to that, the encrypted data is sent via two secure channels, generated by local/remote attestation processes. First the data is pass to enclave E, which then transmits it to enclave F.

5. As the enclave F receives the encrypted data, it gets the key and encrypted value from the encrypted data and stores them in the local database outside the TCB (Trusted Computing Base).

### 4.3.2 Client Retrieves Data From Server

The data retrieval and storage follow the similar procedures in the Vault-SGX system. The former is depicted in Figure 4.6 and their slight differences are related to steps 1 and 3. In step 1, the database receives the message sent by the client. This message includes the data request instruction and the string to indicate the key for which the client wants to retrieve data, e.g., req: {key:
Figure 4.6: The high level overview of data sent from server to client.

'password'). According to the key, the database gets the mapped encrypted data, e.g., `{value: AESenc('1234567')}`, and then sends it to the enclave F for the preparation of data for remote transmitting.

As the data is encrypted by the Encryption Key, in step 3, when the secrets management system receives the encrypted data, it requests the Keyring Key from enclave D and then decrypts it via the AES-NI cipher.
Chapter 5

Implementation

In this chapter, we present the prototype implementation of the system that was introduced in the Chapter 4, Design. Furthermore, the method of our software design process and critical issues are also included in the discussion.

5.1 Prerequisites

Setting up the execution environment for the system is the main preparation required in advance before implementing (and running) the Vault-SGX system. Below we list the the needed tools, deployment, and configuration we have used in order to establish the Vault-SGX system.

- Operating System:
  
  We chose the Ubuntu* 16.04 LTS Desktop 64-bit Linux distribution but Intel SGX as well as Vault would be supported on Windows as well.

- Intel SGX:
  
  According to the installation guide for Linux OS [8], the installers are in three separate packages, which are the Intel(R) SGX driver, Intel(R) SGX platform software (PSW), and Intel(R) SGX SDK. These packages need to be installed and built in order. After the installation, The user can execute the SampleCode under the SGX SDK directory to verify the installation is successful.

- Remote Attestation:
  
  In order to utilize the mechanism of Intel remote attestation, the device needs to register the certification with Intel [23], and then issue a service provider ID (SPID) via the registration. It is the ID of each device when communicating with the Intel attestation server (IAS).
CHAPTER 5. IMPLEMENTATION

- Database (Storage):
  After the encryption of the sensitive data, the encrypted data is stored in the backend storage for use in the future. According to the specification of Vault, it supports plenty of different pluggable storage backends. In Vault-SGX, we chose the Etcd v3 Key-value Datastore, as it supports High Availability. Furthermore, Kubernetes utilizes it as the backend storage, to record the cluster data.

5.1.1 Component Implementation

Although this thesis is an individual stand-alone project, we consider applying suitable software engineering concepts to the implementation process, accounting also for evolutionary development in the future. According to the leading concept of agile software development [11], we aim to divide the system into small components, each of which can be run individually. In addition, we generate the unit tests for each component in order to ensure the component retains the functionality and original purpose after continual improvement and cross function integration.

5.1.2 Local/Remote Attestation

According to the Local/Remote Attestation mechanism of the Intel SGX, before exchanging data between two enclaves, the system utilizes the [9] Diffie-Hellman Key Exchange Library to initialize the protected channel for the data in transit. Intel SGX provides the key exchange library in the application SDK release, which is the trusted library for establishing the attestation process and key exchange protocol. In order to have a clear understanding of the internals of the attestation process, and to be able to utilize it correctly, we base our local/remote attestation implementation on the open source code provided officially by the Intel Software Developer Zone.

5.1.2.1 Local Attestation

In the SDK of Intel SGX, the local attestation process utilizes the Diffie-Hellman Key Exchange APIs, which is described in `sgx_dh.h`, to establish the protected channel between enclaves. Except for the Diffie-Hellman Key Exchange library, we refer the sample code of local attestation provided under Intel SGX SDK directory.

The sample code we reuse provides the example to set up the session between the enclaves. After the session tunnel is ready, the enclave executes the data exchange and the function calling to its counterpart enclave. In this research, we port the test `create_session` and the `test_enclave_to_enclave_call` functionality from the local attestation sample code to the Vault-SGX system, which is the trusted code executed inside the enclave.
In the next chapter, Integration Overview, we introduce how to integrate the local attestation with the other components. The integrated local attestation procedure supports the secure architecture of the whole Vault-SGX system.

5.1.2.2 Remote Attestation

Also the remote attestation utilizes the Diffie-Hellman Key Exchange to create the secure channel for data in transit, similarly to the process of local attestation. In the process of remote attestation, the data structure of the message exchange is described in `sgx_key_exchange.h`. In addition to the key exchange process between the client and the server, the process includes the challenge request toward the third party attestation service, such as Intel Attestation Server (IAS).

According to the remote attestation mechanism, we provide the certificate file and key file of the counterpart, which has been challenged by the service provider. Therefore, upon the request, the SPID of the service provider is also required. In addition, the attestation verification report is encrypted by the RSA-SHA256 algorithm. In order to decode the Verification Report send back by IAS, the certification file of the Attestation Report Signing CA Certificate needs to be included.

5.1.3 Retrieved Encryption Key

In the original unsealing procedure of Vault, when the unsealing process is successful, Vault composes the Master Key and Encryption Key as Keyring Key and then caches the Keyring Key in the memory. However, this might cause data leakage, if the adversary does a cache-attack on the system or get the access to the memory. For this reason, we store the Encryption Key inside the enclave, and then to save it for the future use. In the Figure 5.1, we define two functions for the key storage and key retrieval via an enclave.

```c
enclave {
  trusted {
    public uint32_t putkey([in, size=keysize] uint8_t *keyvalue, uint32_t keylen, uint32_t term);
    public uint32_t getkey([out] uint8_t **keyvalue, [out] uint32_t *keylen, uint32_t *term);
  };
  untrusted {
  }
};
```

Figure 5.1: The trusted code of the encryption key retrieval component.

The enclave is destroyed when the user invokes the `sgx_destroy_enclave` API call or the executing process is dead. In this research, we consider to execute Vault
as a server. When Vault executes the Vault Unsealing, it triggers the enclave created procedure for the Encryption Key storage. Accordingly, unless the server is reset or destroyed, the Encryption Key will be kept inside this enclave as long as the Vault server is alive.

5.1.4 Encrypted/Decrypted in The Enclave

The cryptographic solution in the Intel SGX SDK ports a subset from the Intel IPP Cryptography library. In version 1.7 of the SGX SDK for Linux, Intel SGX adopted the optimized IPP crypto library, which now also includes an AES-NI solution for SGX. However, currently it still only provides the AES-GCM encryption with the key length of 128-bit.

As Vault originally supports AES-NI with a key length of 256 bits, we decided to make our own implementation of the AES algorithm, with AES-NI instructions to support the key length from 128-bit, 192-bit to 256-bit. For the purpose of optimizing the performance, we utilize the NASM assembler to implement the AES-NI cipher with the GCM.

AES-NI is also called Intel Advanced Encryption Standard New Instructions, it is the optimized implementation of the AES-GCM algorithm into 7 new instructions to accelerate the encryption [21] [14]. Following the open source implementations by Go and the Network Security Services (NSS) library, we divide the execution process of AES-NI encryption into 6 functions individually. Next, we list the functions in the execution order and outline the purpose of each function.

- **EXPANDKEYASM**:
  In this function, we utilize the Rijndael key expansion method to expand a set of round keys, the number of the round keys depends on the key length. For instance, the function generates 14 different round keys when the key is 256 bits, as defined in the standard Rijndael algorithm.

- **GCMAESINIT**:
  The function pre-computes the execution element in GHash function, which utilizes the authentication mechanism of the GMAC. As described in the GMAC Section 2.4.1.3 of Figure 2.14, we generate the hash key as the equation (1), and then prepare the Mi execution as the authentication tag.

- **AESEN CBCBLOCK**:
  Implement as the counter mode (CTR) in Section 2.4.1.2. In the depiction of Figure 2.13, which executes the round keys with the nonce (IV) via an encryption procedure, processes the cipher block encryption.

- **GCMAESDATA**:
According to the background discussion of GCM, we implement AEAD (Authenticated Encryption with Additional Data), which is the method to include additional data of the GCM before creating the MAC.

- **GCMAESENC (GCMAESDEC):**
  The function implements the encryption method as shown in Figure 2.11, which depicts encryption of the data with a 128-bit key. The decryption is performed in reverse.

- **GCMAESFINISH:**
  The function implements the last step, depicted in Figure 2.15, which executes the last two polynomial equations in the GHASH function, and then to generate the Auth Tag with the first AES($E_k$) cipher block.

According to the Intel SGX specification, it only supports 64-bit CPUs with Linux. For this reason, we designed to implement the AES-NI with GCM in 64-bit instructions, in a thorough way using NASM (Netwide Assembler).

In the next subsection, we introduce how to link the enclave of SGX with the libraries.

**Using libraries in SGX enclaves**
In the preprocessing step the AES-NI library is imported into the enclave. We create the AES-NI implementation as a shared library (or dynamic-link library (DLL)) in order to link with the enclave. According to the document [18], the Intel SGX forbids the enclave to depend on any shared library. Although the enclave supports to link with the static library, it still causes the critical issue of memory mapping fault when to compile the Go with Cgo enable. For the reason, we utilize the NASM compiler to generate the Assembly code in the object file. It works successfully, the enclave can import the AES-NI function without the exception error.

### 5.2 Integration Overview

In this section, we describe the procedure to integrate the components that we implemented. Furthermore, the tools and solutions that we utilize in the integration process are also in discussed.

#### 5.2.1 Embedded Development with C and Go

The original Vault implementation is written in Go. In the Vault-SGX prototype, the sensitive processes, such as data encryption and key storage, are designed to be executed inside the enclaves and thus be protected by the Intel SGX architecture. As code in the enclaves can only be implemented in C or C++, we utilize Cgo, which is the interface to allow the Go language to interoperate with C libraries.
In our enclave development, we wrap the application in a shared library, which aims to make our enclave implementation flexible to be imported in any program. In addition to declaring the location of the shared library inside the Go program, it has to be exported in the environment variable `LD_LIBRARY_PATH`, which is the solution that makes the system loader able to find the shared library.

### 5.2.2 System Integration

In the original Vault, sensitive data is exposed in the memory without protection. In contrast, in the Vault-SGX system, the sensitive processes are executed inside enclaves, which provide the isolated execution in the protected memory. The sequential flow diagram in Figure 5.2 describes the interaction between enclaves and Vault after we place the sensitive processing inside the enclaves. The diagram is divided into two parts, which are the interaction of Vault server with the KeyStore enclave and the Encryption enclave. Next we follow the diagram step by step and introduce them individually.

![Diagram of Vault embedded Enclave - Sequence of Operations](image)

**Figure 5.2: Vault embedded Enclave - Sequence of Operations.**

- Vault Server interacts with KeyStore Enclave:
The Vault Server uses Cgo to interact with the KeyStore enclave. It calls
the function `store_key` which triggers the system to create the new enclave
if the KeyStore enclave does not exist. After the enclave is ready, param-
eters are passed from the side of Vault Server to store the key inside the
KeyStore enclave. (As the system supports using multiple encryption keys,
the sequential number of the key (term) is provided when storing the key
value inside the enclave.)

After the key storage procedure, the system retains the KeyStore enclave
in the standby mode. I.e., it is pending in the system and waiting for the
requests from the other enclaves. In addition, the KeyStore follows the
lifecycle of the Vault-SGX server, i.e., it is destroyed when Vault-SGX server
is closed, and otherwise it is kept alive in the system.

- **Vault Server interacts with Encryption Enclave:**

  The Vault Server sends the Encrypt request to trigger the Encryption enclave
  initiation. After that, the Encryption enclave creates the session with the
  KeyStore Enclave in order to request the encryption key from the KeyStore
  enclave.

  In this procedure, after the plaintext is encrypted with the encryption key
  inside the Encryption enclave, the session, and the enclave is destroyed in the
  end, as it is unnecessary to keep the Encryption enclave alive. Furthermore,
a smaller EPC can reduce the attack surface in the system.
Chapter 6

Evaluation

The main goal of this thesis is to plug the Intel SGX technology in conjunction with Vault KMS, in order to improve the data security of the cloud platform. In the following security analysis section we list the significant risks related to the adversarial attacks, which can be tackled by Vault-SGX by utilizing the Intel SGX mechanisms. Moreover, the latency and throughput of the system are key performance metrics to measure. In the performance section, we compare the performance of our AES-NI implementation with other existing implementations. Furthermore, the latency of the whole Vault-SGX system is included in the discussion. The above two analyses correspond to the requirements defined in Chapter 3. Last, we introduce suggestions in order to improve the performance and provide the new measurement of the system, in the other issue section.

6.1 Security Analysis

6.1.1 Offline Guessing Attack

As described in the security requirements in Section 3.3, an offline guessing attack happens when the attacker extracts the sensitive data without interaction with the system. In the key management system, the backend storage is the main target of the adversary for executing offline brute force attacks.

In order to prevent the offline guessing attack, the encryption key is used together with the master key as the keyring key, and AES-GCM is used for encrypting the keyring key with the master key before storing into the database. Moreover, The AES cryptographic algorithm provides higher security and is more mathematically efficient than previous cryptography algorithm, such as Data Encryption Standard (DES). In 2001, due to the AES has a benefit of the high complexity and efficiency of the encryption, the National Institute of Standards and Technology (NIST) of the United States published AES as the Federal Information Processing Standard (FIPS) 197 [29], and then use it to protect sensitive
6.1.2 Confidentiality and Integrity of the Sensitive Data

In the upper subsection we introduced AES, which provides confidentiality of the sensitive data. Furthermore, the GCM (Galois Counter Mode) supports authentication tags and thus protects also the integrity of the sensitive data.

In addition the protection of data at rest by encryption, we include the Intel SGX to provide hardware-based protection of the sensitive data in memory, as it provides an isolated execution space for processing the data. Even if, e.g., privileged malware attacks the system or the device, the data can be processed securely in an isolated environment without being exposed to the malware.

6.1.3 Minimizing the TCB

As in the discussion in Section 2.1.1, Intel SGX only consists of the CPU and enclave in the TCB, in order to achieve the small memory footprint (In the depiction of Figure 2.1). As a principle, a smaller attack surface can reduce the risk of exposing sensitive data when the other components are compromised. As we have no right to arrange the CPU, we aim to minimize the size of the enclave as an the primary to reduce the attack surface.

In order to minimize the enclave size, we remove the unnecessary enclaves after the corresponding processes are done. In addition, we reserve a small heap and stack size for each enclave and conduct a prudent of decision the data that is needed to place into the enclaves.

In the evaluation perspective, we utilize the EMMT (Enclave Memory Measurement Tool) to measure the enclave size during the execution of the Vault-SGX system. The enclaves used 12 KB of heap space and 7 KB of stack space when performing key storage and crypto execution inside the enclave.

6.1.4 Other Issues

To respect the security requirements indicated in Section 3.3, the system requires a third party to confirm the trustworthiness of the counterpart devices. According to the remote attestation mechanism, the Intel SGX provides the Intel Attestation Service (IAS), which supports the independent software vendor (ISV) to analyze the counterpart status before exchanging the sensitive data.

In the Design Chapter, we included the remote attestation mechanism in the Vault-SGX prototype. To consider the proper software engineering, we arrange the mechanism as an individual component. At the present stage, the component is executed only individually. In the next stage of the implementation, we plan to integrate the remote attestation part into the Vault-SGX system, in order to
improve the security foundation of the system and support running parts of the system on multiple nodes.

6.1.5 Conclusion

At present, from the security perspective, the goal is to achieve the security requirements of Section 3.3 in the Vault-SGX system with the Intel SGX mechanisms. In the Vault-SGX prototype we have considered the security requirements in the design of the prototype, and accomplished the aims of the requirement. When reviewing of the security of the present Vault-SGX implementation, we also achieve the most of the security requirements. As stated in the Other Issues subsection in the Security Analysis section, the ”Guarantee the trustworthiness of the remote party” is a concept we plan to include in the Vault-SGX implementation in as future work by integrating the remote attestation functionality to the prototype.

6.2 Performance Analysis

Regarding different evaluation metrics, we focus on two factors, namely throughput and latency, when measuring the performance of our AES-NI implementation and the Vault-SGX system. In addition, in order to validate the performance of our solution, we also make a comparison with existing solutions, such as Intel IPP Cryptography and OpenSSL.

The benchmarks are executed in the following environment: Intel NUC KIT NUC7i5BNK, Intel Core i5-7260U Processor, using 16GB DDR4 RAM 2400MHz SODIMM, and running on an Ubuntu 16.04 64-bit desktop OS, with the Intel SGX Linux libraries, release version 2.3.1. Furthermore, all the tests are executed in a single thread with one core processor.

According to the limitation of the enclave page cache (EPC), we scale the data measurement size from 256B to 64MB, as the data size of 64MB reaches the EPC limitation when performing the memory mapping inside and outside the enclave.

6.2.1 AES Cryptography - Throughput

- AESNI-GCM (OpenSSL) vs AESNI-GCM (Shan)

In the depiction in Figure 6.1, the performance of AESNI-GCM (Shan) is 10 times higher than AES-GCM (OpenSSL) when data size reaches 1MB. The throughput of the AESNI-GCM (OpenSSL) increases rapidly when the encrypted data is larger than 16KB. However, after the throughput reaches 2.5 GB/sec at 1MB, the performance has decreased slightly, and then to drops speedily after the data size is 10MB.
CHAPTER 6. EVALUATION

Although the performance drops after the data size at 10MB in the case of the AESNI-GCM (OpenSSL), in the memory monitoring shown in Figure 6.2, it hits 64MB (0.4% of memory space is 64MB, as the memory size is 16G) usage of memory when executed with the 10MB data. Similarly, the performance overhead of AESNI-GCM (Shan) declines when plaintext data raises to 32 MB, whereas the memory allocation includes the size of plaintext and ciphertext when performing the data encryption. According to the memory management and Linux kernel arrangement, we speculate the limitation of 64MB is related to the default value of SWIOTLB (Software Input Output Translation Lookaside Buffer) [12] on the 64-bit platform.

Around the data size of 240KB, the AESNI-GCM implementation of OpenSSL provides better throughput than AESNI-GCM (Shan). According to Figure 6.3, the AESNI-GCM of OpenSSL utilizes three times more memory space than our solution when computing the data encryption in size of 4 MB (In the mathematical calculation, to divide the memory used without encryption, the total amount of memory used by AESNI-GCM of Shan is 8MB, and the memory used by OpenSSL is 24MB).

![Performance Variation and Throughput Comparison](image)

Figure 6.1: The Throughput of AES-GCM (OpenSSL), AESNI-GCM (OpenSSL) and AESNI-GCM (Shan).

- AESNI-GCM (Shan) vs AESNI-GCM with SGX (Shar)

According to the results shown in Figure 6.4, when the data size is less than 4KB, the throughput of AESNI-GCM without SGX obtains a better result than the encryption executed with the protection of enclave. As the memory inside the enclave is encrypted, the data need to swap: the data inside the enclave memory location using an ECALL\(^1\) into the encrypted memory, and then the execution process is protected by the enclave.

\(^1\)It is the function to enter the enclave from the untrusted code. The entry interface
Figure 6.2: The memory used of AESNI-GCM (OpenSSL) with 10MB data size.

Figure 6.3: The memory used of AESNI-GCM (OpenSSL) and AESNI-GCM (Shan) with 4MB data size.

Figure 6.4 shows a sharp change when data is larger than 4KB. The throughput of the encryption with Intel SGX reaches 3 GB/sec when the data size is around 1 MB. However, the encryption without enclave protection only obtains 2.2 GB/sec throughput in the same data size. The difference is related to memory allocation, as we allocate the heap memory in the beginning of the enclave creation. The heap memory provides the constant memory to the enclave for allocation of the memory. However, the unprotected encryption executed without the memory allocation in the beginning requires extra time to find the spare memory and allocation it for the data.

In the enclave configuration we set the maximum heap size to 4GB, which is the upper bound for heap memory reservation in our system. Although

the EPC is higher than the 128MB PRM (Processor Reserved Memory), Intel SGX supports EPC paging, which allows removing the page from the EPC, and then allocating it in the unprotected memory. Even though the pages are outside the protected memory, the EPC management instructions provide the protection to maintain the pages.

The throughput of AESNI-GCM with SGX significantly declines when plaintext size reaches 16MB, as the plaintext and the reply data of ciphertext are all require to swap between the trusted and untrusted zone, in order to achieve the enclave mechanism of data protection. For this reason, it requires four times of the data size to execute the encryption. Similarly, the throughput drops when the memory used reaches 64MB.

- Copied data between trusted bridge vs Cleartext memory without copied data
  - Copied data between trusted bridge:
    Enclave does not share the memory with the untrusted zone of the system. For this reason, the untrusted zone require to utilize the ECALL, in order to allocate the data inside the enclave. Vice versa, when the data inside the enclave, it utilizes the OCALL\(^2\) to swap the data from enclave to the untrusted zone.

  - Cleartext memory without copied data:
    The content of the data is cleartext, the data value does not copy between trusted bridge. The feature is triggered by configure the user_check attribute of the enclave configuration.

According to the depiction in Figure 6.5, the system can process the 5GB of data per second when executing the process inside the enclave. However,

\(^2\)It is the function to enter the untrusted code from the enclave
in this process, we utilize the `user_check` attribute of the enclave configuration, which drops the memory buffer swap into and outside the enclave. Meanwhile, it skips checking the parameter boundary, which might overlap the enclave memory [8].

As in the previous discussion, the throughput drops significantly when the data size reaches 64MB. In comparison with the throughput result of Intel AESNI-GCM, the Intel implementation uses multiple if-else expressions in C, which increases the execution time during the encryption processing.

### 6.2.2 AES Cryptography - Latency

As part of the performance evaluation, we perform latency testing to measure the execution time in each solution. The shown data is the average result after 100 times of execution.

![Throughput: Cleartext Memory without Copied Data](image)

![Throughput: Copied Data between Trusted Bridge](image)

Figure 6.5: The Throughput of AESNI-GCM inside the enclave with/without memory swap.
Figure 6.6: The Latency of AES-GCM (OpenSSL), AESNI-GCM (OpenSSL) and AESNI-GCM (Shan).

- AESNI-GCM (OpenSSL) vs AESNI-GCM (Shan)
  According to Figure 6.6, AESNI-GCM (Shan) provides better performance than OpenSSL. In this particular measurement we do not consider the memory allocation and heap memory size of each process. Thus, the OpenSSL implementation loses the benefit of memory allocation to improve the performance when encrypting large pieces of data.

- AESNI-GCM vs AESNI-GCM with SGX
  In the depiction in Figure 6.7, in the same way as in the throughput measurement in the analysis above, we configure the heap memory in the enclave creation, which improves the performance of the encryption. The reservation of the heap memory provides constant memory, which benefits the memory allocation during the encryption.

In addition, the performance overhead has an obvious difference between AESNI Intel and 256-bit AESNI when the data size increases from 4MB to 16MB. At 4MB, the latency of AESNI Intel is less than AESNI 256-bit. However, the latency of Intel AESNI increases rapidly when the data size reaches 16MB, which is higher than 256-bit AESNI. As an explanation to this observation, we assume the execution time of the boolean expressions inside the Intel AESNI process raises when data size increased.
Figure 6.7: Latency comparison with AESNI-GCM inside and outside the enclave.

<table>
<thead>
<tr>
<th></th>
<th>Vault</th>
<th>Vault SGX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsealing</td>
<td>0.073s</td>
<td>0.115s</td>
</tr>
<tr>
<td>Read</td>
<td>0.021s</td>
<td>0.040s</td>
</tr>
<tr>
<td>Write</td>
<td>0.020s</td>
<td>0.043s</td>
</tr>
</tbody>
</table>

Figure 6.8: Latency comparison with/without the SGX inside Vault system.

### 6.2.3 Vault SGX vs Vault - Latency

We utilize the time command to get statistics of the execution time of vault commands, such as unsealing, read and write. As shown in Figure 6.8, the response time of each test is less than 0.12 seconds, which is less than the average human response time. It means that a human user is unable to recognize the difference between using Vault with or without Intel SGX for in-memory data protection.

### 6.2.4 Other Issues

In addition to the throughput and latency performance characteristics scalability is another major indicator of the performance of the system, which is relevant especially if the system provides the storage backend in the cloud architecture. However, we consider to include the scalability measurement only in the next implementation stage. Furthermore, queuing and multi-process structure are also in consideration, in order to improve the system performance.
6.2.5 Conclusion

The evaluation results show that our AESNI-GCM solution provides a significant performance compared with other AESNI-GCM implementations. In the comparison with the AESNI-GCM introduced by OpenSSL, the small size of data in OpenSSL solution provide better throughput than our solution. However, in the memory monitoring, OpenSSL utilizes three times more memory space than our solution when computing the data encryption in size of 4 MB. Furthermore, we also have better throughput than Intel solution when the data size is bigger than 4KB. In the dropping point of each solution, it has something in common. The throughput provides sharply decline when the real memory used reach 64MB, we speculate the limitation of 64MB is related to the default value of SWIOTLB (Software Input Output Translation Lookaside Buffer) [12] on the 64-bit platform.

In the above conclusion, we list some possible reason to explain the better throughput of our solution. Firstly, we implement the AESNI-GCM in pure assembly language instead of using inline assembly. Secondly, we avoid using boolean expressions to check the format of the passed data before executing the AESNI-GCM algorithm. However, the way to reduce the data format and system status checking may cause unexpected faults without having exception handling in place, and thus we consider it as future work to improve the implementation architecture of the Vault-SGX in this respect.
Chapter 7

Conclusions

In the original concept of the BarbiE [5] (a.k.a. SGX-Barbican) system, utilizing the economic and the scalability benefits of the software-based solution, it uses the Intel SGX technology to replace the HSM component in order to improve the security foundation of the KMS. In this thesis, we apply the similar concept of BarbiE to the Vault KMS, which is an open source software system to manage and protect sensitive data. While the Enterprise version of Vault supports HSM integration, we aim to build the system without depending on the hardware plugin module. Moreover, the capability to prevent both outside and the insider attacks are also required. For this reason, the software-based crypto solution based on Intel SGX technology was chosen as the option to apply to the Vault KMS in this thesis.

Currently, a considerable amount of research work is done related the Intel SGX. It is a state-of-the-art technology in processor security. In this thesis, we propose the Vault-SGX prototype, which extends the Vault KMS crypto architecture by plugging in the mechanisms of Intel SGX, such as attestation and enclaves, which provide the data integrity and confidentiality of the CPU encryption. Moreover, in order to preserve the original security model of Vault, we implemented the 256-bit AES-NI cipher with GCM for execution inside enclaves.

In the evaluation results we compare the performance of our AES-NI implementation with other AES-NI solutions, such as Intel IPP Cryptography and OpenSSL. We measured the throughput with large sized data and executed the data encryption inside/outside the enclave. In conclusion, the Intel SGX crypto plugin for Vault KSM provides an essential enhancement of security as well as performance. Nevertheless, development of the unfinished components of the Vault-SGX prototype, deployment of the KMS in Kubernetes clusters, and the functionality extensions will all continue after this thesis. In the next section, we list the extensions of the Vault-SGX, that we plan to approach in future work.
CHAPTER 7. CONCLUSIONS

Future Work
In this section, we discuss the ongoing processes related to the unfinished components of the Vault-SGX prototype. In addition, the various extensions of the Vault-SGX are also listed in the following subsection.

- **Ongoing Process:**
In the Vault-SGX prototype design, we define a set of system components. However, in the implementation, development of two components are still ongoing due to the time limitation and access to hardware. The components are Remote Attestation and Enclave ID Registry Server. The remote attestation can currently execute only in a standalone mode, and the Enclave ID Registry Server is in the development stage. To use these in conjunction with the completed components is part of the future work on the Vault-SGX prototype.

To consider the deployability requirement of Section 3.4, we plan to improve the availability of the Vault-SGX system in order to achieve the CIA triad model of the information security design. W.r.t. to availability element, the cluster architecture of Kubernetes is the solution to accomplish the deployability requirement, such as failover and load balancing. In the experiment with Vault KMS, we utilize the Vault and Etdc operators\(^1\) to manage clusters deploy to Kubernetes clusters. The integration of the Kubernetes Cluster and the Vault-SGX is included in the future work to achieve the high-availability of Vault-SGX.

- **Extending Vault-SGX to other platforms:**
In the Background Chapter, we introduced also other TEE solutions, provided by the different CPU vendors. In the future work, we intend to port the Vault-SGX architecture to other TEE solutions, such as ARM Trust-Zone. There are, however, differences between SGX and, e.g., TrustZone, which need to be considered. For example, the TrustZone solution does not provide the remote attestation mechanism and the SDK as Intel SGX does. However, some TEE provider they implement its remote attestation solution, such as Trustonic Kinibi [38] and TrustZone-based Integrity Measurement Architecture (TIMA) [34], run within the ARM TrustZone. In our future work, we plan to apply the remote attestation from the other Tee provider with our Vault-SGX architecture, and then to port the whole architecture on the ARM TrustZone.

---

\(^1\)https://coreos.com/blog/introducing-the-etc-operator.html
Bibliography


BIBLIOGRAPHY


[24] JOHN McCARTHY, STANFORD UNIVERSITY. Memorandum to p. m. morse proposing time sharing, January 1959.


BIBLIOGRAPHY


