Encryption Proxies in a Confidential Computing Environment

Mohamad Jamil Al Bouhairi, Mostakim Mullick, Marvin Wolf, Ivan Gudymenko, and Sebastian Clauß

T-Systems Multimedia Solutions GmbH, Riesaer Str. 5, 01129 Dresden, Germany
ivan.gudymenko@t-systems.com

Abstract. With the increasing adoption of cloud native applications, security and privacy are among the pressing concerns hindering a wider adoption of cloud services. One of the main challenges in this context is providing reliable protection of customer data from unauthorized usage, including the cases when adversary gains root access to host systems. Using transparent data encryption, tokenizataion and pseudonymization by deploying an encryption proxy between the end user and cloud provider are some of the established approaches to protect sensitive data in the cloud. In this paper, we extend the advantages of an encryption proxy by deploying and securing it using a shielded execution framework based on Confidential Computing. Our Proof of Concept (PoC) guarantees the necessary security requirements needed to run in an untrusted computing infrastructure. The experimental evaluations show that the PoC achieves reasonable performance results while providing strong security properties with small Trusted Computing Base.

Keywords: Confidential Computing; Cloud Computing; Intel SGX; SCONE; Trusted Execution Environment

1 Introduction

Applications have moved from being a rigid monolithic package to an elastic and loosely connected set of microservices which are often hosted in a cloud infrastructure. The products have been mostly replaced by services that can be subscribed according to one's needs. Thus, a subscriber does not require any bulky infrastructure to host the application on premises but rather a terminal to access the service and a stable network is sufficient. This has certainly proven less hectic as it reduces management cost and time [11]. However, all these benefits come with a hidden cost.

Cloud computing has presented new challenges and problems that an organization must solve to fulfil its privacy and security goals. One of the key issues is data protection. The data is hosted and processed on third party machines and also being served through cloud providers. Several approaches and solutions can be used, but these approaches need to take three core aspects into consideration [6]: data control, usability, and data protection. Data control deals with the management oversight of cryptographic key possession and control that are necessary

for encryption. Usability considers the measurement of how well cloud services and their functionalities remain usable with no restrictions to the end-user despite the data encryption. Finally, data protection involves how companies and organizations control personal data and comply with regulation. Concerns regarding data security and the need to comply with legal regulations have slowed down the process to adopt a cloud-first architecture approach. The use of encryption proxies such as the eperi Gateway [5] have used pseudonymization and encryption methods to transform personal and sensitive data into a form that could not be used in a malicious manner. This insures that *data in transit* and *data at rest* is protected at all times. Third parties with access to Personal Identifiable Information (PII) can be alarming [9]. However, anonymized and correctly pseudonymized data is not observed as PII under the GDPR [10] and therefore are not the subject of data protection. Yet, a cloud application is not able to process encrypted/tokenized data unless it has access to the cryptographic keys. A potential malicious root admin with key access is able to read the data being processed in clear text. Therefore, a practical and secure solution to protect *data in use* is needed.

One emerging technology for confidential computing, trusted execution environments (TEE) [2] (e.g Intel SGX and AMD SEV) have been widely used to deploy secure applications allowing data to be secured in all three states (at rest, in use, and in transit). However, continued adoption of TEE can be impeded by their limited support for unmodified source code implementations. Particularly, in the case of Intel Software Guard Extensions (SGX), the program's source code is expected to incorporate SGX's SDK in order to successfully execute inside of an SGX enclave. This process would entail a lot of effort and is susceptible to performance issues and attacks [7]. Addressing this usability challenge are secure container platforms like SCONE [1] which run the binary code inside SGX while also providing SGX-enabled security guarantees inside cloud environments. The security requirements regarding confidentiality and integrity are achieved by preventing unauthorized parties such as higher privileged system software (e.g OS kernel and hypervisor) and malicious users as mentioned previously to access application data. This ensures that *data in use* is also protected.

It is important that data in all three states are secured. In this paper, we take into consideration all three states and combine both existing technologies to ensure that security guarantees such as confidentiality and integrity are satisfied. Our PoC extends the data privacy techniques accompanied with the eperi Gateway and Intel SGX to design an all-rounded secure approach to data privacy. As per our knowledge, there has been no previous implementation that leveraged the features of a TEE such as Intel SGX in securing the eperi Gateway.

In this paper, we showcase the main contributions of our work as follows:

- We leverage the eperi Gateway's [5] privacy preserving features and extend the privacy preserving requirements with the Intel SGX TEE,
- Using Intel SGX and the secure container platform SCONE, we use a 'lift and shift' approach to transform the encryption proxy to a confidential application,

– We evaluate the efficacy of the proposed proof of concept using benchmarks and compare the overheads as opposed to the native version.

The rest of the paper is organized as follows. Section 2 reviews the related technologies used in this paper. Section 3 provides the implementation scenario and the procedure involved. Section 4 describes the experimental setup. Section 5-6 evaluates the proposed proof of concept with the discussion of the results, and Section 7 concludes the paper.

2 Background and Project Overview

In this section, we briefly review Intel SGX. We describe the existing technologies used in our implementation, SCONE and the eperi Gateway.

2.1 Intel SGX

Intel Software Guard eXtensions[1] (SGX) is a set of extensions to the Intel architecture that provide confidentiality and integrity guarantees for applications even if the underlying operating system or hypervisor is malicious [1, 3]. Launched in 2015, Intel SGX uses secure hardware based TEEs known as *Enclaves*. Enclaves are isolated regions of the memory in which the code running in an enclave is isolated from other untrusted applications including higher privileged ones. Enclave code and data reside in a region of protected physical memory called the enclave page cache (EPC) [1]. Enclave code and data are guarded by CPU access controls while they are in cache. Data in EPC pages is safeguarded at the granularity of cache lines when they are relocated to DRAM. SGX provides EPC of size up to 128 MB in its first generation [12]. The EPC's cache lines written to and fetched from DRAM are encrypted and decrypted by an on-chip memory encryption engine (MEE) [1]. Enclave memory is also integrity protected, i.e., memory alterations and rollbacks are noticed in enclave memory.

In addition, Intel SGX provides *remote attestation* feature. This enables a challenger to verify the integrity of the TEE. The enclave is measured and the corresponding report is signed by the Intel's hardware keys. Combined with the sealing process (encrypting the data before storing on disk), the data confidentiality is ensured on the cloud. As a result, the use of Intel SGX provides confidentiality and integrity of application secrets at all times, despite an adversary that can compromise privileged code running on the hosts of the cloud infrastructure.

2.2 SCONE

Secure CONtainer Environment (SCONE) [1] is a secure container mechanism for Docker that uses the SGX trusted execution support of Intel CPUs to protect container processes from outside attacks. SCONE takes into consideration

[1] https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html

the apparent issues with native Intel SGX SDK development such as exposed container interfaces, memory access and system call overheads and deals with them by providing alternate design suggestions while maintaining small Trusted Computing Base (TCB):

- SCONE implements File System shield to protect confidentiality and integrity of files; Network shield to enable end-to-end TLS encryption; and Console shield to protect the console streams.
- SCONE uses $M : N$ threading scheme while maintaining a thread pool where M application threads are mapped to N OS threads.
- Finally, SCONE enables asynchronous system calls by utilizing the shared memory to pass system call arguments and to collect return values.

In addition, SCONE overcomes the poor memory performance of Intel SGX SDK. Considering the original SGX design goal of protecting tailored code for security-sensitive tasks, Intel provides an SDK to facilitate the implementation of simple enclaves. It consists of an interface definition language with a code generator and a basic enclave library. Unlike SCONE, the SDK lacks the necessary support for system calls and offers only restricted functionality inside the enclave. The use of SCONE allows for better scalability than using Intel SGX SDK alone since it provides efficient thread management by using asynchronous system calls [1].

2.3 Eperi Gateway

Eperi Gateway [5] is an encryption proxy focused on data protection. It can perform a wide variety of tasks like:

- Encryption and pseudonymization of data before they are transferred to the cloud application.
- Internal cryptographic key management handling.
- Enforcing uniform data protection policies across all devices and platforms.

The Eperi Gateway consists of a transparent proxy architecture which can be integrated into any IT environment without requiring the user to apply any major modifications to the underlying infrastructure or existing workflows. The encryption and decryption process occurs transparently and in real-time. Therefore, the solution will run unnoticed in the background while preserving the core functionalities of the cloud and running applications. This process allows the user to have the sole and centralized control over the data's protection. Moreover, the eperi Gateway is flexibly tailored by allowing the client to either use eperi's standard encryption algorithm or one's own algorithm and also it allows the user to choose which fields should be encrypted, tokenized or left as plain text. Configuration data is stored encrypted in an adjacent database. Hence, unauthorized third parties are not able to gain access to the data. However, the threat model does not consider root admins as potential malicious attackers. Therefore, this scenario is specifically considered in this paper.

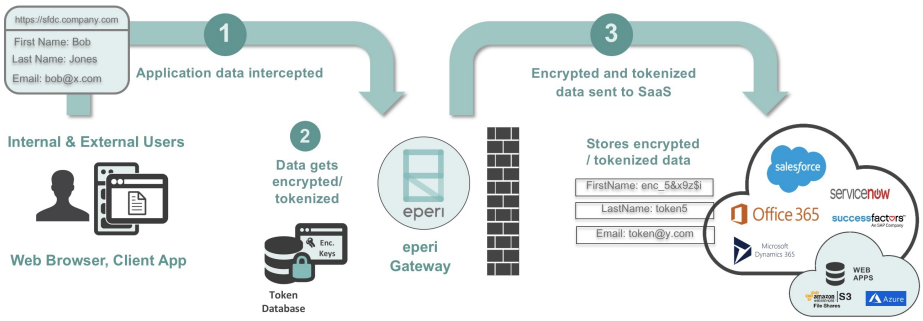

Fig. 1: Eperi Gateway workflow. Taken from [5].

The easiest way to understand the eperi Gateway is to think of it as a "broker" as show in Figure 1. The gateway is separate from the cloud environment whose data must be protected. This decouples cloud applications from cloud data. However, this is for a particular use case. In other use cases, the eperi Gateway can be present on the cloud provider. Depending on the application and access circumstances, it can also act as a reverse, forward, or API proxy.

In essence, the eperi Gateway essentially serves as a forward and reverse proxy. According to Figure 1, users access the eperi Gateway from their network and pass their data to the gateway which is then transparently encrypted before forwarded to the corresponding cloud application. In this way the eperi Gateway acts as a forward proxy. In the opposite direction, the eperi Gateway acts as a proxy server by accepting the data stream that is returned by a cloud service and then decrypts the encrypted data inside it with the cryptographic key already in its possession before forwarding the data to the clients in plain-text.

3 Implementation

The intent of the proof of concept is to secure the instance of a cloud encryption proxy (eperi Gateway) deployed in a cloud environment against the cloud provider and powerful adversaries possessing administration rights on the infrastructure. In this way, we wish to establish a trusted data pipeline in which the customer data is always in a protected state. This means that sensitive data can not be accessed in plain text by entities outside the TCB. Based on this motivation, we propose to leverage an encryption proxy in a confidential environment to realize this trusted pipeline. The proof of concept (PoC) was deployed on a virtual machine with Intel SGX-based CPUs. Microsoft Azure provides Intel SGX enabled confidential compute nodes, i.e, DC Series VMs which are already pre-configured with the required drivers.

3.1 Eperi with SGX

Conventional applications cannot use SGX features in the native form. To convert an application into a confidential application, it has to be modified or cross compiled and built using Intel SGX SDK. However, it is not practical to rewrite the whole application from scratch. Furthermore, Intel SGX SDK supports C/C++ whereas Eperi Gateway is built using Java. We need some other tools to convert this application into a confidential application powered by SGX. One such tool is SCONE which allows us to migrate easily to the confidential environment.

SCONE [1] introduces an efficient lift and shift transformation approach called *Sconification* which involves an automated process using a one step command, *sconify_ image*, to produce the confidential version of the application. SCONE uses a native container image as input, which is created by an existing CI/CD pipeline. The image is converted or "sconified" into a confidential container image that runs inside an enclave where all data and code are protected. SCONE uses an automated single step command to achieve this:

```
sconify_image --from="$NATIVE_IMAGE" --to="$CONFIDENTIAL_IMAGE" ...
```

The *Sconification* procedure consists of the following steps:

1. The command-line tool *sconify_ image* encrypts the service's code and data and copies these encrypted files into the encrypted image.
2. A security policy is created containing metadata and an additional information, e.g, the native image environment and path to working directories, to perform decryption while also checking for file integrity violations. This policy is uploaded to the attested SCONE CAS (Configuration and Attestation Service).
3. To run the confidential container from the encrypted image, SCONE CAS attests the service first to ensure its trustworthiness.
4. If successful, SCONE CAS sends the service secrets specified in the policy in order to ensure the service executes smoothly.

Confidential Application Design. In essence, our confidential application consists of three containers, i.e., *Maria DB*, *Scone LAS* (Local Attestation Service) and *Sconified Eperi*. The LAS handles the local attestation of the enclave. Furthermore, it facilitates the remote attestation by generating a verifiable quote. We aim to deploy the eperi Gateway in a confidential manner using the SCONE platform with all the necessary configuration and the required services running (see Fig. 2). In this manner, the user simply has to run the given script and use the gateway's admin portal normally without any database configurations or source code changes inside the eperi Gateway. The eperi Gateway transparently encrypts the data on its way to the cloud application. Using SGX, the encryption key is secured as opposed to the native eperi Gateway deployment.

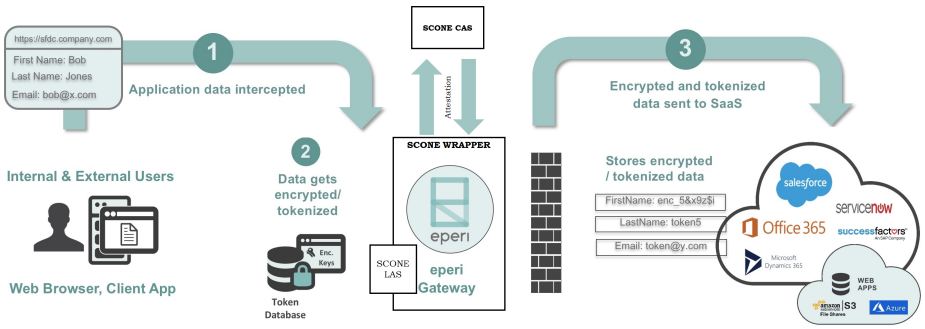

Fig. 2: Sconified eperi Gateway workflow. Adapted from [5].

During the start-up, SCONE attests the eperi container and then starts the bootstrap process for gateway services. The eperi container behaves as a transparent encryption proxy.

The container running MariaDB is not sconified. However, in our setup, MariaDB uses TLS communication while communicating with the eperi Gateway service. This is important as the lack of encryption would introduce security concerns, e.g, Man-in-the-middle attack. In addition, our setup ensures data-at-rest encryption in MariaDB as well. This is ensured through the use of eperi as a key management and encryption plugin with a negligible encryption overhead of around 3-5% [8]. The encryption key uses a 32-bit integer as a key identifier. The eperi Gateway places this key on the key server outside the MariaDB server itself while still remaining in the secure enclave allowing for an additional security guarantee. However, the encryption does not encompass the MariaDB error log, so sensitive information such as PII may be contained in the log as well. Nevertheless, our solution is generic in nature and therefore can be extended to include other confidential applications such as MariaDB. If this path is taken, it would ensure that MariaDB's encryption keys and the TLS certificate are provided only to MariaDB using SCONE's security policy. SCONE CAS enforces this policy, i.e., it ensures that only MariaDB running inside of an enclave will be able to access these keys.

Approach. Sconification simplifies the process of securing a native application to a large extent. Nevertheless, it is not completely straightforward to build a confidential encryption proxy using shielded execution as it demands supporting unaltered applications without degrading the performance. Before the deployment of the sconified image, it is required to compose the native image along with some required configurations.

Some initial configuration was needed to set up the required environment. We followed the eperi documentation [4] to set up a complete work environment and link the eperi Microsoft 365 adapter. The eperi Microsoft 365 adapter was needed to connect to a Microsoft SaaS application running on the cloud provider.

In our case, our backend application was Microsoft Outlook. A custom internet domain has been set up while also the DNS within our Azure cluster has been configured to identify the new domain. The next step is to configure the eperi Gateway and specify the application used. The normal workflow starts by a client sending an email through the eperi Gateway. The eperi Gateway encrypts the sensitive content of the email according to the specified template. If the receiver accesses the Microsoft account through the normal Outlook URL, the contents would be encrypted. If accessed through the custom eperi domain, the contents would appear in plain text.

The first step of converting a native application into a confidential application is to determine the sensitive data. After initial investigation, we determined the directories that contain the application code which is compiled on image creation. The critical directories include the code directory, the data directory and the compiled binary. We cannot make any changes to the configuration inside the eperi Gateway once it is sconified. Therefore, a container with the native eperi base image has been initially created and deployed. Once the container is properly set up, it will be used as the native base image in the sconification process.

After creating a compatible base image and determining the parts to be secured, we can proceed to sconification. It is imperative that the original image supports musl libC or GLibC. In case of the latter, the application binary must be "Position Independent Code" enabled. After setting the environment variables that would be passed after attestation and the proper file system directories to encrypt, we execute the `sconify_image` command using the proper flags. After sconification, we use a normal docker compose YAML file to deploy the prepared image along with MariaDB and SCONE LAS (Local Attestation Service). It is important that we have MariaDB and LAS hosted on the same IPs which we have defined during docker compose configuration in a previous step.

4 Experimental Setup

In this section, we briefly describe the experiments performed for the proposed setup. We based our benchmarks on wrk2 [2] where a web page behind the eperi Gateway proxy is fetched. The HTTP benchmarking tool produces two experimental evaluation metrics: a constant throughput load and accuracy latency details in the high percentiles. The benchmark runs for two minutes using two threads keeping 50 HTTP connections open, and a constant throughput of 100 requests per second. The setup consists of our client load generator, the eperi Gateway with the Microsoft Outlook Exchange under a custom domain as a back-end as shown in the Figure 2. We implemented the services as Docker containers on the Microsoft Azure platform. The Virtual Machine (VM) has 10 vCPUs with 32 GiB of main memory. The disk configuration is irrelevant as all the services fit into memory. We tested two different setups each with different

[2] https://github.com/giltene/wrk2

resource demand. The first setup is the plain native eperi Gateway image (v 21.17.1.0). The image has been pulled from the eperi docker repository with no modifications made. Native means execution that was performed without SGX and therefore also without SCONE. Native executions ran, like the versions using SCONE, inside a Docker container. The performance influence of Docker is therefore out of the equation. The second setup includes the eperi Gateway image converted to a secure image by using the SCONE platform running in an Intel SGX enclave.

Table 1: Average Latency of both setups

Images	**Average Latency (ms)**
Native	423
Sconified	912

5 Results

In this section, we present the results of our experiments on the different setups along with the analysis of the tests.

5.1 Latency

Latency is another key metric to assess the performance of a certain web application as the responsiveness of interactive applications directly influences the quality of user experience. This has become a critical concern for service providers. Latency can be defined as the duration it takes a request to reach its destination across the network and receive an acknowledgement. The average latency of both setups can be visualized in Table 1. The sconified version has an average latency of around twice the native Eperi deployment.

As mentioned in Section 2.3, the above value would outperform a setup using SGX SDK alone. When an enclave performs a system call, SCONE switches to another application thread while the system call is performed by threads running outside the enclave. This minimizes the need for the threads running inside the enclave to exit the enclave. Minimizing the enclave exits is particularly important since it is a costly operation. With this mechanism in place, we can observe a reasonable latency increase.

Figure 3 defines the percentile distribution of latency for HTTP requests from client to Microsoft Outlook via native and sconified deployments of Eperi Gateway. We can ignore the final spike in the graph as an effect of stragglers. Around *one-third* of requests are served within 450 milliseconds and *ninety percent* within 1.2 seconds in sconified version whereas *ninety percent* HTTP requests are served within 500 milliseconds in native deployment of Eperi Gateway. We also have tested out the use of JMeter as a load testing tool. The request

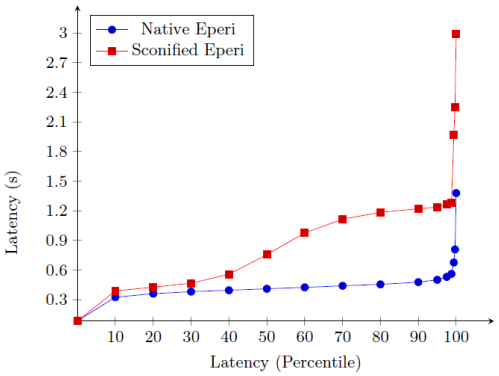

Fig. 3: Latency by percentile distribution

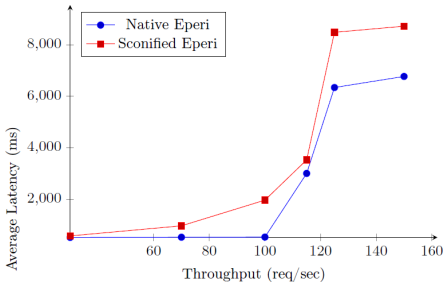

Fig. 4: Throughput versus latency for both setups

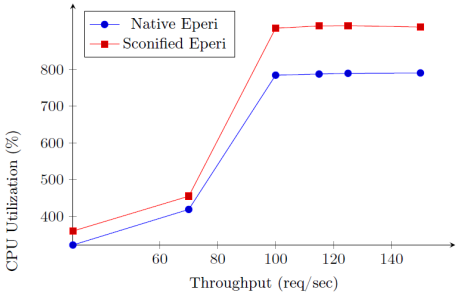

Fig. 5: CPU utilization versus latency for both setups

consisted of random strings in the payload to any increase the size of the request. The request was similar to one used in the previous experiment. The outcome showcased similar results with very minor differences in latency as shown in this section.

5.2 Throughput

The second experimental benchmark consists of issuing requests at increasing constant rates (x-axis) until the response latency spikes (y-axis). Figure 4 shows that both setups exhibit comparable performance until 100 requests per second, at which point the latency of the Sconified deployment increases dramatically. The native eperi deployment performs slightly better, reaching 110 requests per second. A closer look into the CPU utilization shown in Figure 5 explains the aforementioned results. CPU usage was measured using Docker's built-in mechanism for viewing resource consumption, *docker stats*. Both the native and Sconified deployment reach a maximum CPU utilization at 790% and 910%, respectively under maximum throughput.

5.3 Security requirements

In this section, we further look into the non-functional security requirements that are needed to safely run in a public cloud. The cloud provider operates the hardware, the cloud stack, and the OS. Relying on the cloud provider to do all the resource management decreases the complexity of running a cloud application. However, this also forces an application owner to give data and application sovereignty to the cloud provider. Intel SGX supported by the SCONE Platform, however, allows the application owner not to be in a position to "blindly" trust and give this power to neither the cloud provider nor malicious root users. Using SCONE, despite not having full control of neither the hardware nor the software setup, we can ensure that nobody (except for the program itself) can change parts of the program.

The desired level of protection is a design choice made by the application owner. Even if this choice changes, the program does not need to be changed. The objectives considered for this experiment are confidentiality, integrity and consistency. Confidentiality protected means that the protected resource cannot be read by entities not authorized by the security policy of the application. Integrity protected means that the protected resource can only be modified by entities authorized by the security policy of the application. All other changes are automatically detected and cause the program to terminate. Consistency protected means that changing the version of the protected resource will be detected and cause the program to terminate, unless the software update was authorized by the application owner. The assumption is that the program itself and userID are readable but not changeable. This also applies to the secrets and environment variables used by the eperi Gateway.

To verify confidentiality protection of the secrets and environment variables passed by CAS, we observe that all attempts to find values of the passed secrets and environment variables fail, since these variables are confidentiality protected as defined in the policy. However, this is not the case in the native eperi Gateway image where these variables are readable and can be configured. In fact, not even an entity with root access rights to the system of the application owner can access the secrets and environment variables, but only the application itself. To verify integrity protection, we attempt to change the environment variable path by adding another file to read the environment variables from. However, any change to the state of the application gets detected. This can be verified by looking at the hash of the environment variables is identical to the one before modification. To verify consistency protection, we simulate an attack on the consistency protection of the environment variables and check if an older version of these variables has been detected. First, we create and deploy a new version of the eperi Gateway application (version 2) which only differs from version 1 in its environment variables. However, when environment variables from version 2 are uploaded to CAS, the old variables from version 1 will no longer be present in CAS and we therefore have nothing to which we can revert. But for the sake of argument, we can assume that the attacker in some way had indeed got hold of a copy of the version 1 eperi Gateway password. The attacker would have to upload

it to CAS, in order for it to become the correct variable. As soon as the upload to CAS has taken place, however, this is considered to be a new, authorized version (version 3). Hence, we did not succeed in reverting the variables to ones present in version 1 without the change being detected and the attack on the consistency protection failed.

6 Discussion

Confidential computing provides additional security guarantees with respect to confidentiality and data integrity, which is especially relevant in a cloud setting. However, this does not solve all the security related issues. Applications are still vulnerable to threats such as side channel attacks. Nevertheless, if applied in the right way, Confidential Computing can be used to further enhance the security posture of cloud-based applications.

We have seen that Eperi Gateway behaves normally when the whole workflow is tested. This test was performed using the confidential version of the product. Despite certain performance penalties, we could still observe the results allowing for reasonable end user experience. This means that when accessing an application such as Microsoft Outlook with a load that resembles normal user behavior, the confidential eperi Gateway image results in low latencies that resembles values observed when using the native version of the eperi Gateway. Despite one-time bootrapping delays, the overall performance can be considered to be acceptable for the most use case.

7 Conclusion

Cloud Computing introduces new problems regarding data privacy and security as well as data and code sovereignty. We have to trust cloud providers with sensitive information. One solution to protect our sensitive data and code is to encrypt it before uploading it to the cloud, an approach adopted by Eperi. However, we need to secure the environment which hosts the encryption proxy. To achieve this, we use Confidential Computing.

There are different confidential computing technologies one can use. Intel SGX is one of the technologies which reduces our Trusted Computing Base significantly and provides a way to attest our platform albeit with some sacrifice in performance. AMD SEV is another technology which makes it easy to switch to a confidential virtual machine without any changes in application code. However, the TCB in this case includes the whole VM including guest OS.

8 Future Work

We plan to incorporate heavier back-end applications behind the native and confidential eperi Gateway and test their effect on the gateway's behavior. We

also plan to use the confidential image of MariaDB instead of the native version we used in our experiments.

We still cannot generalize the change in performance of third party applications in confidential computing environment. From our observations, we know that there are some added overheads in SGX powered applications. Each application has its own architecture and combined with the technique one might use to convert it into a confidential application, the final architecture could vary greatly. We may try to group the performances of certain architectural and design patterns with probable overheads but that remains to be seen in future.

Confidential Computing is an evolving and dynamically developing field. In VM based trusted execution environments, until now we only had AMD's SEV in the market. However, recently, Intel announced their own VM based TEE, the Trust Domain eXtenstions (TDX) [6]. In addition, Google Cloud Platform offers Secure Encrypted Virtualization (SEV) based confidential computing solutions primarily. We could use Google Kubernetes Engine (GKE) to deploy a cluster of confidential VMs. This can be used to compare the deployment of the confidential eperi Gateway on different types of TEEs.

References

1. Arnautov, S., Trach, B., Gregor, F., Knauth, T., Martin, A., Priebe, C., Lind, J., Muthukumaran, D., O'Keeffe, D., Stillwell, M.L., Goltzsche, D., Eyers, D., Kapitza, R., Pietzuch, P., Fetzer, C.: SCONE: Secure linux containers with intel SGX. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). pp. 689–703. USENIX Association, Savannah, GA (Nov 2016), https://www.usenix.org/conference/osdi16/technical-sessions/presentation/arnautov
2. Consortium, C.C.: Confidential Computing: Hardware-Based Trusted Execution for Applications and Data (01 2021), https://confidentialcomputing.io/whitepaper-01-latest, Last accessed 19 Dec 2021
3. Costan, V., Devadas, S.: Intel sgx explained. Cryptology ePrint Archive, Report 2016/086 (2016), https://ia.cr/2016/086
4. Eperi: Adapter for Microsoft 365, https://adminmanuals.eperi.com/administrator_manuals/en/concepts/egfca_o365_about_document.html, Last accessed 06 Jan 2022
5. Eperi: Eperi gateway: The right approach to effective cloud data protection (06 2018), https://blog.eperi.com/en/eperi-gateway-the-right-approach-to-effective-cloud-data-protection, Last accessed 19 Feb 2022
6. Intel: Intel®Trust Domain Extensions (2020), https://www.intel.com/content/dam/develop/external/us/en/documents/tdx-whitepaper-v4.pdf, Last accessed 22 Feb 2022
7. Mahhouk, M., Weichbrodt, N., Kapitza, R.: Sgxometer: Open and modular benchmarking for intel sgx. In: Proceedings of the 14th European Workshop on Systems Security. p. 55–61. EuroSec '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3447852.3458722, https://doi.org/10.1145/3447852.3458722
8. MariaDB: Eperi Key Management Encryption Plugin, https://mariadb.com/kb/en/eperi-key-management-encryption-plugin/, Last accessed 18 March 2022

9. Shakil, Arif, M., Sohail, S.S., Alam, M.T., Ubaid, S., Nafis, M.T., Wang, G.: Towards a two-tier architecture for privacy-enabled recommender systems (pers). In: Wang, G., Choo, K.K.R., Ko, R.K.L., Xu, Y., Crispo, B. (eds.) Ubiquitous Security. pp. 268–278. Springer Singapore, Singapore (2022)
10. Skendžić, A., Kovačić, B., Tijan, E.: General data protection regulation — protection of personal data in an organisation. In: 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). pp. 1370–1375 (2018). https://doi.org/10.23919/MIPRO.2018.8400247
11. Srivastava, P., Khan, R.: A review paper on cloud computing. International Journal of Advanced Research in Computer Science and Software Engineering **8**, 17 (06 2018). https://doi.org/10.23956/ijarcsse.v8i6.711
12. Xing, B., Shanahan, M., Leslie-Hurd, R.: Intel® Software Guard Extensions (Intel® SGX) Software Support for Dynamic Memory Allocation inside an Enclave. pp. 1–9 (06 2016). https://doi.org/10.1145/2948618.2954330