Trusted Client-Side Encryption for Cloud Storage

Marciano da Rocha, Dalton Cézane Gomes Valadares, Angelo Perkusich, Kyller Costa Gorgonio, Rodrigo Tomaz Pagno, and Newton Carlos Will

1 Department of Computer Science, Federal University of Technology - Paraná, Dois Vizinhos, Brazil
marciano@alunos.utfpr.edu.br, {rodrigopagno,will}@utfpr.edu.br

2 Department of Mechanical Engineering, Federal Institute of Pernambuco, Caruaru, Brazil
dalton.valadares@caruaru.ifpe.edu.br

3 Department of Electrical Engineering, Federal University of Campina Grande, Campina Grande, Brazil
perkusic@dee.ufcg.edu.br

4 Department of Computer Science, Federal University of Campina Grande, Campina Grande, Brazil
kyller@dsc.ufcg.edu.br

Abstract. Nowadays, users are delegating the data storage to cloud services, due to the virtually unlimited storage, change history, broadband connection, and high availability. Despite the benefits and facilities, it is necessary to pay extra attention to data confidentiality and users’ privacy, as numerous threats aim to collect such information in an unauthorized manner. An approach to ensure data confidentiality is the use of client-side encryption, with the user taking control of the encryption keys and defining which files or data will be encrypted. This scheme is already explored by many applications on personal computers and also as a native feature in some smartphone operating systems, but are still susceptible to certain types of attacks. Aiming to improve the security of the client-side encryption approach, we propose to apply the Intel Software Guard Extensions (SGX) to perform data sealing, creating a secure vault that can be synchronized with any cloud storage service, while relying on the SGX to protect the key handling. To validate our proposal, we build a proof of concept based on the Cryptomator application, an open-source client-side encryption tool specially designed for cloud storage services. Our results show an overall performance better than the original Cryptomator application, with stronger security premises. Thus, our solution proved to be feasible and can be expanded and refined for practical use and integration with cloud synchronization services.

Keywords: Intel SGX · Data sealing · File encryption · Confidentiality · Integrity · Secure storage · Cloud storage
1 Introduction

Data confidentiality is a central concern today, since the users are increasingly using cloud storage services to keep their files and access them from any device connected to the Internet. These files can contain sensitive data, such as financial and medical information, which makes governance and data security extremely important [38]. Following this trend, there is an increase in the cyber-attacks that aim to explore sensitive information from people and companies [26,31,51]. Companies faced an average of 25 new threats per day in 2006, up to 500,000 in 2016 [61], and 61% of the CEOs are concerned with the state of the cybersecurity of their company [42].

When the user delegates data storage to a cloud provider, the security and privacy of such data must gain special attention, since a third party can get access to these files and information, and the confidentiality of sensitive data must be ensured. Cloud storage providers offer some mechanisms to protect the data, such as access control and encryption, but even so, data leaks are not uncommon [1,37,53]. Also, these storage providers do not always guarantee or respect the privacy of their users [8,15,19].

To add an extra layer of security, the user can perform data encryption before sending them to the cloud storage provider. This can be done by encrypting single files, the entire volume, or using secure vaults, ensuring the confidentiality of all sensitive data to be synchronized with a cloud service. This approach also puts the user in the control of the encryption keys, avoiding that a data leakage in the cloud provider compromises them.

Despite the user having total control of the data and the encryption procedure, this approach also brings some extra concerns. First of all, major encryption solutions use a password defined by the user to derive the encryption key, and such password is often weak and can be discovered using dictionary attacks, default passwords, rainbow tables, or even brute force [44]. Another factor is that commonly file and disk encryption solutions store the encryption key in the main memory and, even when the user chooses a strong password, a malicious application can recover the encryption key by performing a memory dumping or similar techniques.

Hardware-based mechanisms, such as Intel Software Guard Extensions (SGX) [28], can be used to generate and protect the encryption keys. SGX allows the creation of an encrypted area of memory to protect code and data from unauthorized access, even when the operating system or hypervisor is compromised. It also provides sealing, enabling the sensitive data to be securely stored in the permanent storage, without the need to handle the encryption keys [4].

In this context, we seek to improve the security of traditional file/disk encryption systems, while decreasing the user’s need to trust the cloud service provider.

1.1 Contribution

In this paper, we propose the use of Intel SGX features to perform data sealing and processing more securely, since these operations are performed only inside
an enclave, which is supposed to be protected even against adversaries with high privileges inside the operating system. We implemented, as a proof of concept, an application using the Cryptomator, an open-source client-side encryption tool used for the protection of cloud files. We modified the Cryptomator, integrating it with an Intel SGX application, to use the SGX data sealing feature, which is performed only inside an enclave. This way, the data encryption/decryption processes are performed more securely, extending this security also for the keys, since they are also handled inside the enclave.

The main contributions of this work are:

- The proposal to improve the security of the disk/file encryption process using Intel SGX;
- The implementation of such proposal considering an Intel SGX application integrated with the Cryptomator tool;
- The performance evaluation of a proof of concept, considering six different hardware combinations.

This paper is an extended version of our previous work entitled Secure Cloud Storage with Client-Side Encryption Using a Trusted Execution Environment [46], which has been presented at the 10th International Conference on Cloud Computing and Services Science (CLOSER). We expanded our former work with new performance results and comparisons, and a more accurate security assessment on the Intel SGX technology and our prototype. Moreover, we present a reviewed and expanded set of related works, also covering aspects of using Intel SGX in cloud environments.

1.2 Outline

The remainder of this paper is organized as follows: Sect. 2 provides background information about the cloud storage and disk encryption techniques, as well the Intel Software Guard Extensions technology; Sect. 3 presents previous research closely related to our proposal, in the field of cloud storage security, the use of Intel SGX in file encryption and in cloud storage services; Sect. 4 describes the solution proposed by this paper and the prototype developed in order to validate the proposed approach; Sect. 5 presents the performance evaluation of the prototype on six different hardware combinations; the threat model as well the security evaluation of the Intel SGX technology and the proposal solution are presented in Sect. 6; the limitations of our solution are described in Sect. 7; future work are described in Sect. 8 and, finally, Sect. 9 concludes the paper.

2 Background

This Section presents a brief description of important concepts used in this work, such as cloud storage, file/disk encryption, the Cryptomator software and Intel Software Guard Extensions.
2.1 Cloud Storage

Currently, due to the advance in networking technology and broadband communication, a wide range of systems are using cloud providers to store configuration and application data. Besides, users are using cloud storage services to keep personal files and data, as well as companies are delegating the storage of financial and commercial information also to the cloud. Cloud storage brings many benefits, such as virtually unlimited storage, version history for each file, and access to data at any time from any device connected to the Internet.

As the use of these services grows, the concern with the confidentiality of the stored data also becomes a central point, since these services become targets of cyber-criminals [54]. Encryption is largely used to ensure the confidentiality of sensitive data, and its techniques can be applied in two manners: client-side and server-side.

Server-side encryption delegates the encryption procedures and key management to the cloud storage provider. This approach can provide the isolation of users’ data, but there are several risks of keeping the encryption key and data encrypted with it in the same place. Furthermore, the cloud provider still has access to users’ files in plaintext.

On the other hand, in client-side encryption, the user is responsible for executing the encryption and handling the encryption keys, being a user-centric approach. This way, the data are sent to the cloud already encrypted, ensuring that the cloud provider cannot access the plaintext information, and the user can choose which data will be encrypted. Also, the user can store the encryption key in a token device, or derive it from a password.

Finally, both client-side and server-side encryption can be combined to provide extra layers of security to sensitive data.

2.2 File and Disk Encryption

Nowadays, many people and companies store a range of sensitive data to their cell phones and computers, which must have confidentiality ensured. Such data include personal documents, financial information, passwords, medical information, among others. All of this information may be susceptible to third party access, either physically via the device itself, or through network access, such as public WiFi connections at airports and other places. Setting a lock password in the device may be not sufficient to protect these data from unauthorized access. Thus, the use of file and disk encryption, such as Full Disk Encryption (FDE) mechanisms, became an important mechanism to ensure data confidentiality.

FDE aims to encrypt entire volumes and can be divided into two categories: hardware-based FDE and software-based FDE. In hardware-based FDE, the encryption is performed by the disk controller, which also handles the encryption keys. Besides, the MBR (Master Boot Record) is also encrypted, preventing manipulation attacks [36]. While avoiding a range of attack vectors with key manipulation, some implementations allow the stored data to be extracted from the drive without knowing the key used to perform the encryption [34].
On the other hand, software-based FDE intercepts all the operating system (OS) requests and encrypts the data before storage, or fetches encrypted data on the storage media and returns OS-readable information. The CPU performs the encryption/decryption, and the main memory stores the encryption keys. Commonly, these solutions use AES (Advanced Encryption Standard) as the default encryption algorithm [36]. Software-based encryption solutions are also frequently used to encrypt single files and to create secure vaults, which consists of an encrypted folder or a virtual file system where the sensitive files are stored.

### 2.3 Cryptomator

Cryptomator is a solution specially developed for encrypting files that are synchronized with cloud storage services, performing client-side encryption. It enables creating vaults locally, in a virtual hard drive, and syncing them to the cloud storage service. All files inside these vaults are encrypted individually in a transparent way, before being sent to the cloud. This approach ensures that the user’s data are never unencrypted in the cloud and allows the cloud storage provider to maintain a file update history. Cryptomator also provides a directory structure obfuscation, ensuring higher data confidentiality [21].

The application is developed in Java and uses FUSE to create a file system in userspace. Cryptomator is divided into three main modules: CryptoFS, CryptoLib and Graphical User Interface (GUI). The CryptoFS implements the virtual file system, enabling the vault creation and offering a communication interface between the operating system and the encryption procedures. The CryptoLib provides encryption/decryption functions, being invoked by CryptoFS to perform these operations. Finally, the GUI enables the users to access their vaults.

The procedure performed by Cryptomator to read the file data, presented in Fig. 1, can be described in 12 steps:

1. The user requests the operating system to open a file;
2. The operating system requests FUSE for file data. FUSE allows that the userspace applications export a filesystem to the Linux kernel, with functions to mount the file system, unmount it and communicate with the kernel;
3. The FUSE forwards this request to the Cryptomator, using the CryptoFS library;
4. The Cryptomator requests the operating system to have the file data fetched from the storage device;
5. The operating system locates the data;
6. These data are loaded into the main memory;
7. The operating system provides these data to the Cryptomator;
8. The CryptoFS library sends the encrypted data to the CryptoLib library;
9. The CryptoLib library decrypts the received data and returns them to CryptoFS library;
10. The CryptoFS library sends the decrypted data to FUSE;
11. FUSE forwards such data to the operating system;
12. Finally, the operating system provides the user with the decrypted file.
The CryptoFS splits the file into blocks with 32 KB to send to the CryptoLib module, instead of encrypting the whole file at once. The data writing procedure is quite similar, with the plaintext file being sent first to the CryptoLib module before being written in the storage device. All the files stored in the vault can be synchronized with a cloud storage service, in its encrypted form, ensuring also the data confidentiality over the network.

2.4 Intel Software Guard Extensions Technology

Intel Software Guard Extensions (SGX) is an extension of the Intel CPU instruction set, that aims to protect sensitive information in an application, ensuring that these data cannot be modified or accessed by software running at a higher privilege level. The new instructions enable the developer to define private regions of memory, called enclaves, blocking any access to these data from outside the enclaves. The main goal of the SGX technology is to reduce the Trusted Computing Base (TCB) to a piece of hardware and software, as shown in Fig. 2.

The SGX blocks unauthorized accesses to data or code inside an enclave, generating access faults. The data inside the enclave memory are encrypted using a 128-bit time-invariant AES-CTR algorithm, and the CPU stores the encryption key, without access by external entities [5,29]. During the data transmission between the registers, the internal access control mechanisms, from the processor, avoid unauthorized access. Malware and even system code with higher privilege can’t change the data inside the enclave, ensuring their confidentiality and integrity.

Intel SGX also provides mechanisms to detect if any part of the enclave was tampered with. Each enclave has an author self-signed certificate that enables
it to prove that it was correctly loaded to the memory and is trustworthy. The enclave is linked to the application that created it, which is open to any inspection and analysis [33].

To extend the data confidentiality and integrity primitives to a permanent storage device, Intel SGX provides the sealing feature. This mechanism allows the enclave to request a sealing key, which is an encryption key derived from the CPU and the enclave signature, and is used to seal the data before storing them. This key never goes out the boundaries of the CPU, and there is no need to store it for recovering (unseal) the data. The AES-GCM algorithm is used for data sealing and the AES-CMAC algorithm is used to generate the sealing key [4,5].

Enclaves are also able to share data with each other by using an attestation mechanism, which allows an enclave to prove that it is legitimate, has not been tampered with and was loaded correctly, allowing the creation of a secure channel for communication. Local attestation is used when both enclaves are running in the same platform, defining a symmetric key by a Diffie-Hellman key agreement procedure, authenticated by the hardware. This procedure ensures that both enclaves were loaded correctly and that all static measurements are valid. Remote attestation is also provided [4], allowing that a third-party ensures the application really runs on an SGX.

3 Related Work

This Section contains related works that also consider cloud storage security and privacy, disk encryption using the Intel SGX technology, and the use of Intel SGX in cloud storage.
3.1 Cloud Storage Security and Privacy

The security and privacy of files stored in the cloud are a central concern well explored in the literature. Zhou et al. [64] proposed trusted models to address the trustworthiness of cryptographic Role-Based Access Control (RBAC) schemes. With these trusted models, users can decide to join a specific role and decide what data will be accepted. Also, Wang et al. [59] proposed a scheme that combines Ciphertext-Policy Attribute-Based Encryption (CP-ABE) and blockchain to create a decentralized framework that provides full control to the data owner.

Different frameworks have been proposed aiming to ensure data security and privacy. An approach relies on a Trusted Third Party (TTP) to provide data confidentiality [22]. The TTP service performs encryption/decryption features and can be employed in the users’ machine or on top of the cloud storage service. Another work uses a data slicing approach to achieve secure data storage in an enterprise environment, allowing the migration of application data to cloud environments while keeping the confidentiality [55]. Decomposition is used to split and store data in different cloud storage providers [7,41], while client-side AES encryption is used to increase data security and confidentiality [6].

Hardware-based security mechanisms are also applied to provide security and privacy in cloud storage services. Crocker and Querido [20] proposed an approach that uses a hardware token and the OAuth 2.0 protocol to build a two-factor encryption architecture, which can be used with any cloud storage provider. Valadares et al. [56] proposed an architecture that requires TEEs to decrypt and compute the sensitive data, in cloud/fog-based IoT applications, and applies authentication and authorization for the participants, increasing the security and privacy of the data.

3.2 File and Disk Encryption with Intel SGX

The use of Intel SGX to provide data confidentiality and integrity is explored in the literature. Karande et al. [30] proposed an approach to ensure the integrity of operating system log files by using data sealing, avoiding unauthorized modification. Condé et al. [16] also applied data sealing to improve the security of user authentication procedures in operating systems, ensuring the confidentiality of credential files.

In a generic way, Richter et al. [45] proposed the isolation of kernel modules into enclaves, preventing the spreading of vulnerabilities from one module to others, which could compromise the system. To validate the proposal, the authors created a Loadable Kernel Module (LKM) that registers a new mode inside the kernel encryption API, allowing the use of enclaves to perform disk encryption. Esteves et al. [25] also presented a stackable file system framework that uses the isolated execution provided by Intel SGX to extend the SafeFS architecture, a FUSE-based framework that enables the construction of secure distributed file systems. Data sealing is also combined with FUSE to protect in-memory and persistent storage [11].
Finally, Ahmad et al. [2] combined Intel SGX and Oblivious RAM (ORAM) to create a secure file system resistant to side-channel attacks. The file system runs in an enclave and all the requests are made by other application enclaves, by using a secure channel created through the attestation procedure. Another approach aims to create a tamper-resistant storage using a two-way authentication mechanism to transfer data from the SSD firmware to the SGX enclave, in a secure way [3].

3.3 Cloud Storage with Intel SGX

Intel SGX technology is also used to provide secure access control on cloud storage environments. Peterson et al. [40] used the remote attestation process to ensure data confidentiality and integrity in processing and storage, adding a protective layer that enables privacy assurance and access control. Contiu et al. [17] presented a cryptographic access control with zero knowledge, which used the Intel SGX to address the impracticality of the Identity-Based Broadcasting Encryption (IBBE), reducing the computational complexity. Another approach implements a cryptographic access control by using Anonymous Broadcast Encryption (ANOBE) scheme with Intel SGX, achieving scalability by using micro-services [18]. Finally, Djoko et al. [24] proposed a stackable file system approach that provides confidentiality and integrity, by using a client-side SGX enclave to handle the encryption keys.

4 Proposal

This work proposes the use of the sealing feature, provided by Intel SGX, to perform client-side encryption on files that will be synchronized with a cloud storage provider. This approach aims to enable strong data confidentiality and integrity properties, by using a Trusted Execution Environment (TEE), while keeping the user in control of the files. To this purpose, we chose to modify the Cryptomator application (Sect. 2.3), which is a modular and open-source solution for client-side file encryption, working with a wide range of cloud storage services.

Cryptomator is designed to support multiple encryption modes without any changes in the main architecture of the software. The CryptoLib module describes a set of methods that must be implemented to perform the encryption and decryption procedures. The original Cryptomator implementation brings only an AES encryption mode that uses default Java functions to perform the operations.

Our implementation consists of adding a new encryption mode, performing the operations within an enclave, and using the sealing key, generated by the CPU. We implement all the functions required by the CryptoLib, allowing the user to choose between the AES encryption or the SGX option. Since the enclave code is developed in C++ language, we use the Java Native Interface (JNI) to mediate the communication between the CryptoLib and the enclave.
Figure 3 presents the communication between the CryptoFS and CryptoLib modules. We can see the two encryption modes supported by the modified CryptoLib. When using the SGX mode, CryptoLib will communicate with the SGX enclave to send the data blocks. The shaded box indicates the SGX enclave, which manipulates the key and encrypts and decrypts the data blocks, in a trusted way. The communication flow is explained below:

Fig. 3. Workflow performed for data encryption and decryption through the Cryptomator with the inclusion of data sealing feature provided by Intel SGX technology [46].

1. The CryptoFS library calls the encrypt/decrypt functions from the CryptoLib library, which contains AES and SGX modes;
2. In the SGX mode, the CryptoLib library creates an enclave and sends to it each data block to perform the encryption/decryption procedure;
3. The encrypted/decrypted blocks are sent back to the CryptoLib;
4. The encrypted/decrypted data are forwarded to the CryptoFS library.

5 Performance Evaluation

To evaluate the proposal based on the implemented proof of concept, we ran a set of performance tests in different hardware combinations, as well we compared our solution with the unmodified Cryptomator application and other disk/file encryption solutions.

5.1 Experimental Setup

We run tests in a custom PC with the following settings: motherboard with a Z390 chipset, 16 GB 2666 MHz RAM, SGX enabled with 128 MB PRM size, running Ubuntu 16.04.6 LTS, kernel 4.15.0-51-generic. We used the Intel SGX SDK 1.7 for Linux and set the stack size at 4 KB and the heap size at 1 MB.

To measure the CPU contribution to the performance impact of the encryption task, we used two distinct CPUs:
- **CPU 1:** Dual-core 3.8 GHz Intel Pentium G5500;
- **CPU 2:** Octa-core 3.6 GHz Intel i7 9700K.

We also used three distinct storage devices to perform the tests, each one containing different characteristics regarding performance for data reading and writing, as described below:

- **Device 1:** HDD Samsung ST1000LM024, 1 TB storage size, 5400 RPM spin speed, 145 MB/s maximum data transfer rate, 8 MB buffer DRAM size;
- **Device 2:** SSD SanDisk PLUS, 240 GB storage size, 530 MB/s sequential read, 440 MB/s sequential write;
- **Device 3:** SSD NVMe M.2 Samsung 970 EVO Plus, 250 GB storage size, 3500 MB/s sequential read, 2300 MB/s sequential write.

For all devices, we used an Ext4 filesystem partition. To achieve the highest read and write rates for each device, we used a RAMDisk as data destination and source for the read and write evaluations, respectively. Thus, there is no speed limitation by the source or destination of the data.

### 5.2 Methodology

The performance tests were carried out using five different solutions to read and write data:

- **Without Encryption:** Data read and write operations on the storage device without any type of encryption;
- **LUKS:** Data read and write operations on the storage device using native cryptographic mode in Ubuntu with LUKS (Linux Unified Key Setup);
- **VeraCrypt:** Data read and write operations on the storage device using VeraCrypt with encrypted containers;
- **Cryptomator:** Data read and write operations on the storage device using the original Cryptomator application;
- **Cryptomator-SGX:** Data read and write operations on the storage device using the implemented solution, which integrates the SGX data sealing with the Cryptomator application.

We choose LUKS [10] to perform the comparison since it is an in-kernel solution and it presents a low overhead. Also, VeraCrypt [27] is a multiplatform solution that uses a compiled binary to obtain high performance.

Two distinct scenarios were considered: the copy of a single file, and the copy of multiple files. We used the ISO image of the CentOS 7 [12] operating system to evaluate the transfer of a single file. Featuring the transfer of several files, we extracted the same image, using the recursive directory transfer mode, and used its files and folders. The RSync tool was used to transfer data.

To obtain stable results, Intel TurboBoost, SpeedStep, and HyperThread extensions were disabled. Besides, we executed the tests ten times for each scenario, clearing the system’s cache before each execution, aiming to reduce variations from the operating system.
5.3 File Writing Performance

The first test aims to evaluate the performance of a single file writing. Figure 4 shows the results obtained by the different storage devices, when using the Intel Core i7 CPU. Figure 5 presents the results for the Intel Pentium CPU.

Analyzing the results, we can note that LUKS and VeraCrypt, which use multithreading programming, achieve higher transfer rates, very close to the rates shown without any encryption feature. Since Cryptomator uses a single thread approach, its data transfer rate is lower than the other applications, mainly in...
storage devices with higher throughput. This characteristic is accentuated with the Intel Core i7 CPU use. In Intel Pentium CPU, the Cryptomator application achieves higher transfer rates, due to the higher processing clock. Besides, the difference between it and the other file encryption solutions is reduced, due to the smaller number of processing cores.

Our solution (Cryptomator-SGX) demonstrated higher transfer rates than the vanilla Cryptomator in this scenario, considering all six combinations of CPU and storage device. The better performance is achieved by the use of CPU instructions to perform the encryption, and by the optimized code in the SGX enclave. These optimizations suppress the overhead imposed by the data transfer operations through the JNI interface, the context changes, and communication with the enclave.

In the second scenario (writing several files of different sizes), we seek to identify the impact caused by the file indexing operations, performed by the file system to build the references to the files. Figure 6 shows the results in this scenario when using the Intel Core i7 CPU. We observe a performance drop, both in vanilla Cryptomator and in our solution, when compared to LUKS and VeraCrypt. Comparing the vanilla Cryptomator and the Cryptomator-SGX implementations, we can see that there is no significant difference in performance.

![Fig. 6. Transfer rate writing multiple files with Intel Core i7 9700K CPU.](image)

Figure 7 shows the results obtained using the Intel Pentium CPU for the same scenario. We can also notice that both Cryptomator and Cryptomator-SGX achieve a higher transfer rate when compared to the Intel Core i7 CPU. Comparing our solution with the original Cryptomator implementation, the results show a sensitive drop, especially in the M.2 storage device.
5.4 File Reading Performance

We also evaluated the overhead in reading operations. Figure 8 shows the result in the first scenario, reading a single file, using the Intel Core i7 CPU. In this scenario, we can see that the vanilla Cryptomator application achieves transfer rates close to or higher than LUKS, despite its limitations. Also, the Cryptomator-SGX solution achieves better performance in all the storage devices, when compared to the original Cryptomator application. This performance improvement is mainly due to the use of CPU decryption instructions, instead of the Java functions.

Fig. 8. Transfer rate reading a single file with Intel Core i7 9700K CPU.
Likewise, Fig. 9 presents the results in the same scenario using the Intel Pentium CPU. Our solution also achieves transfer rates very close to or better than the vanilla Cryptomator application. In the SSD and M.2 storage devices, our solution increases the transfer rates achieved by Cryptomator by \(1.56 \times\). When using the SSD storage device, we can notice even better performance than LUKS.

In the second scenario, reading multiple files, our solution achieved good performance, mainly when compared with the original Cryptomator implementation. Figure 10 shows the result with the Intel Core i7 CPU. Our solution achieves higher transfer rates than the vanilla Cryptomator across all the storage devices. In the M.2 storage device, we can also see a better performance than LUKS.
Figure 11 presents the results for the same scenario, but using the Intel Pentium CPU. We can also note that our solution achieves higher transfer rates than the original Cryptomator implementation, except when using the SSD storage device. Also, in the HD and SSD storage devices, our solution presents results very close to LUKS.

![Graph showing transfer rates](image)

**Fig. 11.** Transfer rate reading multiple files with Intel Pentium G5500 CPU.

Finally, the overhead presented by our solution, when compared to VeraCrypt, is mainly due to the single thread architecture applied in the Cryptomator application. The effects of this design decision stand out mainly when performing the reading of multiple files, with these requests executed in a serialized way.

### 6 Security Assessment

This section presents the threat model, the security analysis of Intel SGX technology, and the security assessment of the implemented solution.

#### 6.1 Threat Model

Our threat model considers that the adversaries aim to access confidential data stored on the computer’s hard disk. To do this, they can have physical access to the computer, and can even remove the disk and install it on another machine. Doing this, they can use higher computational power to apply techniques to retrieve the password used in the key generation, or the encryption key itself. We also assume that they have installed some malicious software on the user’s computer, trying to obtain the encryption key by a memory dumping, or the vault password by using a keylogger.
The adversaries can also monitor the network traffic, to obtain data transmitted between the user’s computer and the storage provider. The storage provider may be deployed in an untrusted cloud, or even may be compromised. They can also obtain access to the storage provider and retrieve data, or the storage provider can leak its encryption keys or the data in plaintext.

6.2 Intel SGX Security Assessment

As described in Sect. 2.4, the main goal of Intel SGX is to keep the Trusted Computing Base (TCB) very small, being composed only by a small portion of hardware and software. This approach presents a reduced attack surface and raises the security level of the applications. Data inside the enclave are kept safe even when the operating system or the hypervisor is compromised.

Intel SGX aims to ensure data and code confidentiality and integrity when running inside enclaves. To achieve this, all the enclave data and code are put inside a protected region of memory, encrypted by a 128-bit AES-CTR algorithm, with the encryption key being randomly generated on each boot. Also, instructions and data are decrypted only by the CPU, and the encryption key never goes out its boundaries. The SGX sealing feature provides confidentiality and integrity for data stored outside the enclave, using a 128-bit AES-GCM algorithm. The encryption key is derived from the CPU and the enclave signature, and also never goes out the CPU boundaries [5].

One important point is that the development environment must be secure, otherwise, the enclave code can be compromised. In order to build secure enclaves, the API and SDK software must be installed from the official channels, keeping them updated with their most recent versions. Also, the developer must take precautions when manipulating data inside the enclave, to avoid leakage of sensitive data.

Despite the security mechanisms employed by the Intel SGX technology, it does not cover side-channel attacks. Wang et al. [60] presented four side-channel attack vectors that SGX offers no protection: power statistics, cache miss statistics, branch timing, and page accesses via page tables. Data access patterns and physical attacks against the CPU, such as fault injection or reprogramming of machine code functionalities, are also not covered by the SGX threat model [9,13].

Many works in the literature explore side-channel attacks in SGX applications [32,35,48,50,58], as well as propose several solutions to mitigate such attacks. Software-based solutions are commonly achieved through the use of Intel Transactional Synchronization Extensions (TSX) [14,49] and Oblivious RAM (ORAM) [2,43,47]. Bulck et al. [57] presented an analysis of the vulnerabilities and mitigations in shielding runtimes implementations, including SGX.

6.3 Cryptomator-SGX Security Assessment

To assess the security of our solution, we consider that the Intel SGX technology works properly, according to its specifications, and that the proposed
solution development environment is reliable. We also consider that the Intel SGX technology is secure, focusing our validation only on the changes made in the Cryptomator.

The Intel SGX sealing feature allows using a CPU derived key to encrypt data and ensures that the key can only be retrieved in the same platform. Thus, when the data are copied to another machine, it will not be possible to recover the key used to perform the decryption, even using the enclave that encrypted the data, as the derivation of the key also depends on the CPU. The sealing operation is performed by a 128-bit AES-GCM algorithm, and even an attacker copying the data to a high-performance environment, the encryption breaking procedure will be very expensive, taking a long time. Also, to ensure that only the data owner has access to them on the platform where they were encrypted, we kept a second layer of security, in which the user defines a password to access the vault. This password is also used to encrypt the file names.

In case an attacker installs some malicious software to obtain the sealing key, through memory dumping or similar techniques, they will not be successful, since that key never leaves the CPU, being derived by it when necessary. Also, if an attacker has physical access to the machine, or successfully cracks the sealing key, they still need to recover the vault password or key, in order to obtain access to file names and the vault itself. Another approach is to implement a solution that uses the CryptoLib module to bypass the Cryptomator security mechanisms, but they will still need to input the vault password or key. This approach provides a second layer of security to the user’s data.

On the other hand, if the cloud storage provider is compromised and leak the user data, or if attackers have access to the cloud infrastructure or cloud service, they will still need the vault key to retrieve the file names and the sealing key to retrieve the file data. This security level is ensured by the client-side encryption performed by our solution. The server-side encryption, performed by the cloud storage provider, will add an extra layer of security to the user data and will represent a new obstacle for an attacker to overcome.

Finally, if attackers listen to the network traffic, to obtain the plaintext data or some tip about the encryption keys, they will not have any success. All the data sent to the cloud storage provider are encrypted by the client, and no plaintext data or key are sent.

7 Limitations

The main limitation of our solution relies on the single thread architecture of the Cryptomator application, which results in lower performance in multi-core platforms, as shown in Sect. 5. We decided to add the SGX sealing feature without making changes to the Cryptomator architecture, to maintain the compatibility with the original project. Thus, only one enclave is created to perform the encryption/decryption procedures, and the operations are queued by the Cryptomator, which reduces the performance when compared with multithread solutions. A multithread version of the solutions requires major changes in the Cryptomator architecture and may be explored in future work.
Due to a limitation of the operating system, the sealing feature cannot be applied to encrypt the file names. Then, we keep the Cryptomator default encryption feature for this purpose. Thus, the user is still required to input a password to open the vault and generate an encryption key to cipher the file names. This limitation also adds an extra layer of protection in the solution: since the SGX sealing feature performs an authenticated encryption by using the AES-GCM algorithm, additional information is added to the encrypted data. As a result, the file size is increased by around 1.7%.

Besides, the data are only accessible in the machine in which they were sealed since the sealing key is derived from the CPU. To transfer the data to another machine, the remote attestation procedure can be added to the solution, to establish a secure channel between the two machines. Using this secure channel, data can be transferred to the destination machine, where they can be sealed with the key generated in this machine. In case the CPU used in the data encryption is lost, due to a malfunction or other occurrence, the data cannot be unsealed.

File sharing through the cloud storage provider also becomes difficult, because of the nature of the sealing key. This limitation can be circumvented with the remote attestation mechanism, or by using another CPU-independent key derivation scheme for file encryption. Since the data are encrypted in the client-side, cloud deduplication techniques may be ineffective; this situation is addressed by Yan et al. [63]. Finally, our solution does not ensure data confidentiality on the I/O devices. Dhar et al. [23], Peters et al. [39], and Weiser and Werner [62] present solutions to this problem.

8 Future Work

The use of data sealing, provided by the Intel SGX technology, has proven to be an efficient solution for encrypting sensitive files. However, due to the constant need to perform backups and remote access to data, the current solution can be improved to use the remote attestation feature, allowing the transfer of data from the containers between two machines, which are running the application through secure channels. To provide strong security guarantees, it is also possible to seal the Cryptomator configuration data. Such change requires adjustments to the CryptoFS library structure, which may remove compatibility with the main project.

The Intel SGX technology can also be applied to manipulate the encryption keys used by the cryptographic algorithms running within an enclave, removing the dependency of the CPU key. For this, it is necessary to use the same concept applied by Richter et al. [45]. In this scenario, security guarantees provided by the Intel SGX technology prevent an attack on the primary memory from obtaining the user’s password or the key used by encryption, reducing the chances of a successful attack to allowing data migration to another platform.
9 Conclusion

In this paper, we proposed using the data sealing feature, provided by the Intel SGX, to ensure data confidentiality at the client-side, allowing the users to synchronize their sensitive data with cloud storage services in a secure way. We developed a proof of concept based on the Cryptomator application, a file encryption solution specially designed to work with a wide range of cloud storage services.

Experimental results demonstrated that, in most cases, our solution performs better than the original Cryptomator implementation, by using Intel SGX features and specific CPU instructions to perform data encryption/decryption. Compared with other solutions, our results show that significant changes are necessary for the Cryptomator design to obtain better transfer rates.

Our solution also provides an extra level of security, as discussed in Sect. 6, by combining the data sealing feature with a user-defined password to manage the vaults. Since the main memory does not store the sealing key, and all the sensitive operations are carried out inside the enclave, the probabilities of data or key leakage are reduced.

Finally, our solution proved to be feasible and extensible to be applied in many contexts. The source code is available at https://github.com/utfpr-gprsc/cryptomator-sgx.

References

1. Data breach report: cloud storage exposes 270,000 users’ private information (2020). https://www.securitymagazine.com/articles/91985-data-breach-report-cloud-storage-exposes-users-private-information
2. Ahmad, A., Kim, K., Sarfaraz, M.I., Lee, B.: OBLIVIATE: a data oblivious file system for Intel SGX. In: Proceedings of the 25th Network and Distributed System Security Symposium. Internet Society, San Diego (2018). https://doi.org/10.14722/ndss.2018.23284
3. Ahn, J., et al.: DiskShield: a data tamper-resistant storage for Intel SGX. In: Proceedings of the 15th Asia Conference on Computer and Communications Security. ACM, Taipei (2020). https://doi.org/10.1145/3320269.3384717
4. Anati, I., Gueron, S., Johnson, S., Scarlata, V.: Innovative technology for CPU based attestation and sealing. In: Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy. ACM, Tel-Aviv (2013)
5. Aumasson, J.P., Merino, L.: SGX secure enclaves in practice: security and crypto review. In: Proceedings of the Black Hat. Black Hat, Las Vegas (2016)
6. Babitha, M., Babu, K.R.R.: Secure cloud storage using AES encryption. In: Proceedings of the International Conference on Automatic Control and Dynamic Optimization Techniques, pp. 859–864. IEEE, Pune (2016). https://doi.org/10.1109/ICACDCT.2016.7877709
7. Branco Jr., E.C., Monteiro, J.M., Reis, R., Machado, J.C.: A flexible mechanism for data confidentiality in cloud database scenarios. In: Proceedings of the 18th International Conference on Enterprise Information Systems, pp. 359–368. SciTePress, Rome (2016). https://doi.org/10.5220/0005872503590368
8. Branscombe, M.: Has Microsoft been looking at user files to find the 75tb OneDrive hoarders? (2015). https://www.techradar.com/news/internet/cloud-services/has-microsoft-been-looking-at-user-files-to-find-the-75tb-onedrive-hoarders--1308186
9. Brasser, F., Müller, U., Dmitrienko, A., Kostiainen, K., Capkun, S., Sadeghi, A.R.: Software grand exposure: SGX cache attacks are practical. In: Proceedings of the 11th USENIX Workshop on Offensive Technologies. USENIX, Vancouver (2017). https://www.usenix.org/conference/woot17/workshop-program/presentation/brasser
10. Broz, M.: Linux Unified Key Setup (2020). https://gitlab.com/cryptsetup/cryptsetup/wikis/home
11. Burihabwa, D., Felber, P., Mercier, H., Schiavoni, V.: SGX-FS: hardening a file system in user-space with Intel SGX. In: Proceedings of the 10th IEEE International Conference on Cloud Computing Technology and Science. IEEE, Nicosia (2018). https://doi.org/10.1109/CloudCom2018.2018.00027
12. CentOS: The CentOS Project (2020). https://www.centos.org/
13. Chen, G., et al.: Racing in hyperspace: closing hyper-threading side channels on SGX with contrived data races. In: Proceedings of the 39th IEEE Symposium on Security and Privacy. IEEE, San Francisco (2018). https://doi.org/10.1109/SP.2018.00024
14. Chen, S., Zhang, X., Reiter, M.K., Zhang, Y.: Detecting privileged side-channel attacks in shielded execution with Déjà Vu. In: Proceedings of the ACM on Asia Conference on Computer and Communications Security, pp. 7–18. ACM, Abu Dhabi (2017). https://doi.org/10.1145/3052973.3053007
15. Clover, J.: Hackers using iCloud’s find my iPhone feature to remotely lock macs and demand ransom payments (2017). https://www.macrumors.com/2017/09/20/hackers-find-my-iphone-remote-mac-lock/
16. Condé, R.C.R., Maziero, C.A., Will, N.C.: Using Intel SGX to protect authentication credentials in an untrusted operating system. In: Proceedings of the 23rd Symposium on Computers and Communications. IEEE, Natal (2018). https://doi.org/10.1109/ISCC.2018.8538470
17. Contiu, S., Pires, R., Vaucher, S., Pasin, M., Felber, P., Réveillère, L.: IBBE-SGX: cryptographic group access control using trusted execution environments. In: Proceedings of the 48th Annual International Conference on Dependable Systems and Networks, pp. 207–218. IEEE, Luxembourg City (2018). https://doi.org/10.1109/DSN.2018.00032
18. Contiu, S., Vaucher, S., Pires, R., Pasin, M., Felber, P., Réveillère, L.: Anonymous and confidential file sharing over untrusted clouds. In: Proceedings of the 38th Symposium on Reliable Distributed Systems, pp. 21–2110. IEEE, Lyon (2019). https://doi.org/10.1109/SRDS47363.2019.00013
19. Cox, J.: Hackers stole account details for over 60 million Dropbox users (2016). https://www.vice.com/en_us/article/nz74qb/hackers-stole-over-60-million-dropbox-accounts
20. Crocker, P., Querido, P.: Two factor encryption in cloud storage providers using hardware tokens. In: Proceedings of the Global Communications Conference Workshops. IEEE, San Diego (2015). https://doi.org/10.1109/GLOCOMW.2015.7414154
21. Cryptomator: Cryptomator system architecture (2019). https://cryptomator.org/security/architecture
22. Dahshan, M., Elkassas, S.: Framework for securing data in cloud storage services. In: Proceedings of the 11th International Conference on Security and Cryptography, pp. 267–274. SciTePress, Vienna (2014). https://doi.org/10.5220/0005043802670274

23. Dhar, A., Puddu, I., Kostiainen, K., Capkun, S.: ProximiTEE: hardened SGX attestation by proximity verification. In: Proceedings of the 10th Conference on Data and Application Security and Privacy, pp. 5–16. ACM, New Orleans (2020). https://doi.org/10.1145/3374664.3375726

24. Djoko, J.B., Lange, J., Lee, A.J.: NeXUS: practical and secure access control on untrusted storage platforms using client-side SGX. In: Proceedings of the 49th Annual International Conference on Dependable Systems and Networks, pp. 401–413. IEEE, Portland (2019). https://doi.org/10.1109/DSN.2019.00049

25. Esteves, T., et al.: TrustFS: an SGX-enabled stackable file system framework. In: Proceedings of the 38th International Symposium on Reliable Distributed Systems Workshops. IEEE, Lyon (2019). https://doi.org/10.1109/SRDSW49218.2019.00012

26. Huang, K., Siegel, M., Madnick, S.: Systematically understanding the cyber attack business: a survey. ACM Comput. Surv. 51(4) (2018). https://doi.org/10.1145/3199674

27. IDRIX: VeraCrypt - free open source disk encryption with strong security for the paranoid (2020). https://www.veracrypt.fr/en/Home.html

28. INTEL: Intel Software Guard Extensions Programming Reference (2014). https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf

29. Intel: Intel Software Guard Extensions SDK for Linux OS Developer Reference. Intel Corporation (2016). https://01.org/sites/default/files/documentation/intel_sgx_sdk_developer_reference_for_linux_os.pdf

30. Karande, V., Bauman, E., Lin, Z., Khan, L.: SGX-Log: securing system logs with SGX. In: Proceedings of the Asia Conference on Computer and Communications Security. ACM, Abu Dhabi (2017). https://doi.org/10.1145/3052973.3053034

31. Khraisat, A., Gondal, I., Vamplew, P., Kamruzzaman, J.: Survey of intrusion detection systems: techniques, datasets and challenges. Cybersecurity 2 (2019). https://doi.org/10.1186/s42400-019-0038-7

32. Kim, D., et al.: SGX-LEGO: fine-grained SGX controlled-channel attack and its countermeasure. Comput. Secur. 82, 118–139 (2019). https://doi.org/10.1016/j.cose.2018.12.001

33. McKeen, F., et al.: Innovative instructions and software model for isolated execution. In: Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy. ACM, Tel-Aviv (2013). https://doi.org/10.1145/2487726.2488368

34. Meijer, C., van Gastel, B.: Self-encrypting deception: weaknesses in the encryption of solid state drives. In: Proceedings of the 40th Symposium on Security and Privacy, pp. 72–87. IEEE, San Francisco (2019). https://doi.org/10.1109/SP.2019.00088

35. Moghimi, A., Eisenbarth, T., Sunar, B.: MemJam: a false dependency attack against constant-time crypto implementations in SGX. In: Smart, N.P. (ed.) CT-RSA 2018. LNCS, vol. 10808, pp. 21–44. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-76953-0_2

36. Müller, T., Freiling, F.C.: A systematic assessment of the security of full disk encryption. IEEE Trans. Dependable Secure Comput. 12(5), 491–503 (2015). https://doi.org/10.1109/TDSC.2014.2369041
37. Muncaster, P.: Verizon hit by another Amazon S3 leak (2017). https://www.infosecurity-magazine.com/news/verizon-hit-by-another-amazon-s3/

38. Onwujeke, G., Thomas, M., Osei-Bryson, K.M.: Using robust data governance to mitigate the impact of cybercrime. In: Proceedings of the 3rd International Conference on Information System and Data Mining. ACM, Houston (2019). https://doi.org/10.1145/3325917.3325923

39. Peters, T., Lal, R., Varadarajan, S., Pappachan, P., Kotz, D.: BASTION-SGX: Bluetooth and architectural support for trusted I/O on SGX. In: Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy, pp. 1–9. ACM, Los Angeles (2018). https://doi.org/10.1145/3214292.3214295

40. Peterson, R., et al.: Vallum: privacy, confidentiality and access control for sensitive data in cloud environments. In: Proceedings of the 11th International Conference on Cloud Computing Technology and Science. IEEE, Sydney (2019). https://doi.org/10.1109/CloudCom.2019.00026

41. Pottier, R., Menaud, J.: Privacy-aware data storage in cloud computing. In: Proceedings of the 7th International Conference on Cloud Computing and Services Science, pp. 405–412. SciTePress, Porto (2017). https://doi.org/10.5220/0006294204050412

42. PwC: global economic crime survey 2016: adjusting the lens on economic crime. Technical report, PwC (2016). https://www.pwc.com/gx/en/economic-crime-survey/pdf/GlobalEconomicCrimeSurvey2016.pdf

43. Richter, L., Götzfried, J., Müller, T.: Isolating operating system components with Intel SGX. In: Proceedings of the 1st Workshop on System Software for Trusted Execution. ACM, Trento (2016). https://doi.org/10.1145/3007788.3007796

44. Richter, L., Götzfried, J., Müller, T.: Isolating operating system components with Intel SGX. In: Proceedings of the 1st Workshop on System Software for Trusted Execution. ACM, Trento (2016). https://doi.org/10.1145/3007788.3007796

45. da Rocha, M., Valadares, D.C.G., Perkusich, A., Gorgonio, K.C., Pagnino, R.T., Will, N.C.: Secure cloud storage with client-side encryption using a trusted execution environment. In: Proceedings of the 10th International Conference on Cloud Computing and Services Science, pp. 31–43. SciTePress, Prague (2020). https://doi.org/10.5220/0009130600310043

46. Sasy, S., Gorbunov, S., Fletcher, C.W.: ZeroTrace: oblivious memory primitives from Intel SGX. In: Proceedings of the Network and Distributed System Security Symposium. Internet Society, San Diego (2018). https://doi.org/10.14722/ndss.2018.23239

47. Schwarz, M., Weiser, S., Gruss, D., Maurice, C., Mangard, S.: Malware guard extension: abusing Intel SGX to conceal cache attacks. Cybersecurity 3(1) (2020). https://doi.org/10.1186/s42400-019-0042-y

48. Shih, M.W., Lee, S., Kim, T., Peinado, M.: T-SGX: eradicating controlled-channel attacks against enclave programs. In: Proceedings of the Network and Distributed System Security Symposium. Internet Society, San Diego (2017). https://doi.org/10.14722/ndss.2017.23193

49. Shinde, S., Chua, Z.L., Narayanan, V., Saxena, P.: Preventing page faults from telling your secrets. In: Proceedings of the 11th Asia Conference on Computer and Communications Security, pp. 317–328. ACM, Xi’an (2016). https://doi.org/10.1145/2897845.2897885
51. Singh, M., Singh, M., Kaur, S.: Issues and challenges in DNS based botnet detection: a survey. Comput. Secur. 86 (2019). https://doi.org/10.1016/j.cose.2019.05.019
52. Sobchuk, J., O’Melia, S., Utin, D., Khazan, R.: Leveraging Intel SGX technology to protect security-sensitive applications. In: Proceedings of the 17th International Symposium on Network Computing and Applications. IEEE, Cambridge (2018). https://doi.org/10.1109/NCA.2018.8548184
53. Spring, T.: Insecure backend databases blamed for leaking 43 TB of app data (2017). https://threatpost.com/insecure-backend-databases-blamed-for-leaking-43tb-of-app-data/126021/
54. Sumathi, M., Sangeetha, S.: Survey on sensitive data handling—challenges and solutions in cloud storage system. In: Peter, J.D., Alavi, A.H., Javadi, B. (eds.) Advances in Big Data and Cloud Computing. AISC, vol. 750, pp. 189–196. Springer, Singapore (2019). https://doi.org/10.1007/978-981-13-1882-5_17
55. Trang, T.T.X., Maruyama, K.: Secure data storage architecture on cloud environments. In: Proceedings of the 11th International Joint Conference on Software Technologies, pp. 39–47. SciTePress, Lisbon (2016). https://doi.org/10.5220/0005974400390047
56. Valadares, D.C.G., da Silva, M.S.L., Brito, A.E.M., Salvador, E.M.: Achieving data dissemination with security using FIWARE and Intel software guard extensions (SGX). In: Proceedings of the 23rd Symposium on Computers and Communications. IEEE, Natal (2018). https://doi.org/10.1109/ISCC.2018.8538590
57. Van Bulck, J., Oswald, D., Marin, E., Aldoseri, A., Garcia, F.D., Piessens, F.: A tale of two worlds: assessing the vulnerability of enclave shielding runtimes. In: Proceedings of the Conference on Computer and Communications Security, pp. 1741–1758. ACM, London (2019). https://doi.org/10.1145/3319535.3363206
58. Van Bulck, J., Piessens, F., Strackx, R.: SGX-Step: a practical attack framework for precise enclave execution control. In: Proceedings of the 2nd Workshop on System Software for Trusted Execution, pp. 4:1–4:6. ACM, Shanghai (2017). https://doi.org/10.1145/3152701.3152706
59. Wang, S., Wang, X., Zhang, Y.: A secure cloud storage framework with access control based on blockchain. IEEE Access 7, 112713–112725 (2019). https://doi.org/10.1109/ACCESS.2019.2929205
60. Wang, W., et al.: Leaky cauldron on the dark land: understanding memory side-channel hazards in SGX. In: Proceedings of the 24th ACM SIGSAC Conference on Computer and Communications Security. ACM, Dallas (2017). https://doi.org/10.1145/3133956.3134038
61. Weafer, V.: Report: 2017 threats prediction. Technical report, McAfee Labs (2016). https://www.mcafee.com/au/resources/reports/rp-threats-predictions-2017.pdf
62. Weiser, S., Werner, M.: SGXIO: generic trusted I/O path for Intel SGX. In: Proceedings of the 7th Conference on Data and Application Security and Privacy, pp. 261–268. ACM, Scottsdale (2017). https://doi.org/10.1145/3029806.3029822
63. Yan, H., Li, X., Wang, Y., Jia, C.: Centralized duplicate removal video storage system with privacy preservation in IoT. Sensors 18(6) (2018). https://doi.org/10.3390/s18061814
64. Zhou, L., Varadharajan, V., Hitchens, M.: Trust-based secure cloud data storage with cryptographic role-based access control. In: Proceedings of the 10th International Conference on Security and Cryptography, pp. 62–73. SciTePress, Reykjavik (2013). https://doi.org/10.5220/0004508600620073