Regular Paper

Towards Resistance to Memory Inspection Attacks on Plausibly Deniable Distributed File Systems

Ryouga Shibazaki†, Hiroshi Inamura‡, and Yoshitaka Nakamura*

†Graduate School of Systems Information Science, Future University Hakodate, Japan
‡School of Systems Information Science, Future University Hakodate, Japan
* Faculty of Engineering, Kyoto Tachibana University, Japan
{g2120017, inamura}@fun.ac.jp, nakamura-yos@tachibana-u.ac.jp

Abstract - Data protection has become an important issue in Internet services. In storage systems, conventional methods such as full disk encryption are generally used, but this alone cannot protect against forced attacks of key disclosure. PDE (Plausibly Deniable Encryption), which enables the denial of the existence of confidential information, has been proposed, and by disclosing the decoy key, it has become possible to protect the user from the force to disclosure the key. It is an issue to be considered that the main memory is attacked at runtime due to the use in the cloud and the spread of virtualization technology. Therefore, we are proposing PTEE FS that realizes an encrypted file system using the concept of PDE in a trusted execution environment (TEE). To provide the resistance to exploiting the knowledge from the use of disclosed the decoy key, we introduce FID unification mechanisms. Regarding the performance of PTEE FS, we will evaluate the estimated performance given by the overhead of using TEE by using a model that imitates actual use on the cloud and using file synchronization between the server and client as the actual use model on the cloud.

Keywords: Plausibly Deniable Encryption, OS Security, Trusted Execution Envirionment.

1 BACKGROUND

Leakage of confidential data related to privacy endangers the privacy of data owners and leads to the loss of social credibility of the leaked organization, so protection of such data has become an important issue. Traditional methods such as full disk encryption are commonly used in storage systems, but these methods make it difficult to maintain confidentiality when access to computer hardware or administrator privileges is stolen by an attacker. On the other hand, Plausibly Deniable Encryption (PDE), which is a new concept of encryption, has been proposed [1]. PDE protects confidential information sufficiently by allowing the existence of information to be denied. By disclosing the decoy key, PDE protects against the extortion of the decryption key by an attacker. While admitting that the encrypted file system exists in the system, the attacker is given the decoy key to access the decoy area, but the existence of the hidden area and its contents are kept secret. From the perspective of storage system configuration, PDE's existing research primarily protects sensitive information in persistent storage, and it is assumed that the main memory, which controls the existence of confidential information at runtime, will not be attacked. As an attack on the main memory, a memory inspection attack is assumed in this paper. This is an attack that illegally takes a snapshot of the main memory and obtains confidential information.

According to the white paper of the Japanese Ministry of Internal Affairs and Communications [2], it can be seen from Fig. 1 that data storage and backup are performed using the cloud. In this way, cloud services have become widespread due to the spread of virtualization technology in recent years, and attacks on main memory have become an issue to be considered. There is a risk that someone who understands the system configuration attempts to attack the main memory with a vulnerability on the premise of using technology that controls hardware privileges such as virtualization technology and hypervisor used in cloud services. An example of incidents involving suspected administrative compromise is the illegal access to data for about 100 million people stored on Amazon Web Services at Capital One, an American bank, in 2019 [3].

With the development of virtualization technology, hardware support functions are being incorporated into CPUs to be able to perform processing that guarantees data confidentiality even when such privileged users and terminal administrators are not credible [4].

So far, the purpose of this study is to construct an encrypted file system that is resistant to attacks not only on the persistent storage device but also on the main memory and that can deny the existence using the concept of PDE. We proposed a system using Intel SGX as a hardware-protected execution an environment in the realization of an encrypted file system [5].

In this paper, we examine countermeasures against attacks that are established on the premise that the attacker knows the existence of the decoy key and decoy data for PTEE FS (PDE with Trusted Execution Environment File System). In Chapter 4, we describe the problems of an attack using knowledge from the use of disclosed decoy key. First, introduce a PDE system and threat analysis as assumptions for our previous proposal. Then, we explain an attack using knowledge from the use of disclosed decoy key, which is a threat to be solved in this paper. Chapter 5 we propose FID (file ID) unification, which is a method for solving attacks using knowledge from the use of disclosed decoy key explained in Chapter 4. The basic concept, procedures, and integrated structure of the hidden and

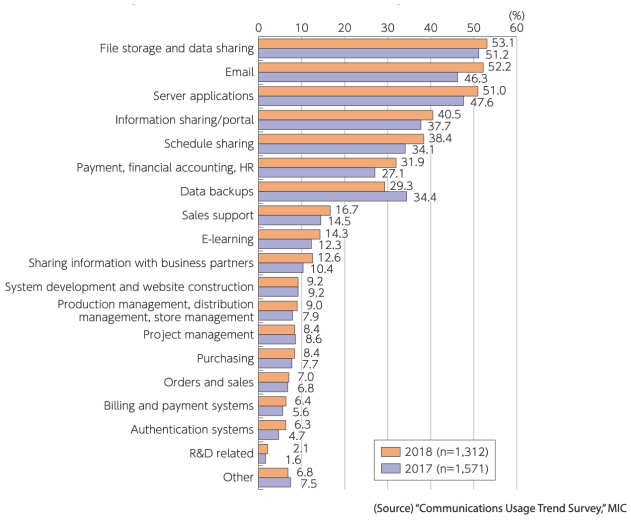

Figure 1: Breakdown of cloud service usage in enterprises

decoy file will be presented. In Chapter 6 and 7, as an evaluation of the processing time in normal access of PTEE FS, a model that imitates the actual use on the cloud is used, and the performance is evaluated in consideration of the overhead due to the use of TEE. In addition, as the processing time of the program started on-demand, the performance is evaluated in consideration of the additional latency due to calling the FID merge processing described in Chapter 5. In the evaluation of processing time in normal access, file synchronization between server and client is used as an actual usage model on the cloud. In the evaluation of the processing time of the program started on-demand, the local file created by referring to the existing research [6] by Leung et al. is used.

2 RELATED RESEARCH AND RELATED TECHNOLOGY

This section describes the concept of Plausibly Deniable Encryption, its application to file systems, and Intel SGX, which is being examined for application to the realization of attack resistance to main memory.

2.1 Plausibly Deniable Encryption

Plausibly Deniable Encryption (PDE) was proposed by Canetti et al. [1] as one of the encryption methods. Traditional disk encryption methods including full disk encryption have the problem that they cannot be protected if the owner is forced to disclose the decryption key by an attacker. Therefore, PDE, which was proposed as one of the methods to protect the owner from the key disclosure extortion attack, enables the protection attack by using the decoy key. PDE is a characteristic of using a decoy key, which enables protection from key disclosure extortion attacks. As shown in Fig. 2, PDE applies special encryption to confidential information that can be decrypted with both a decoy key and a private key, unlike conventional encryption. Decryption with the decoy key gives the decoy plaintext, and decryption with the private key gives the original plaintext. When the legitimate user is attacked by

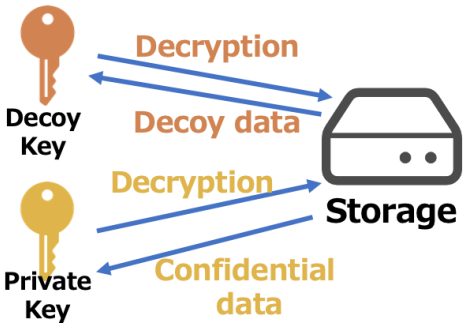

Figure 2: Overview of PDE

an attacker forcing key disclosure, the user can give the decoy key to the attacker. Since the attacker thinks that the decoy key is the original private key, it allows the original confidential information unnoticed and be kept secret.

On the other hand, the disadvantage is that the size of the ciphertext becomes extremely large, which may make the attacker suspicious of applying a special cipher. Furthermore, traces of confidential information may be obtained from the file system and the physical storage medium layer, etc., and considering these, it cannot be said to be a practical method. However, the idea of PDE is that the decoy key gives decoy information and the private key gives the confidential information that can be used.

2.2 Applications of the PDE Concept

Using the idea of PDE, a method was proposed to bring confidentiality using two types of techniques, steganography and hidden volume, instead of using simple encryption. First, a PDE method using the concept of steganography was proposed by Anderson et al. [7] and Chang et al. [8]. The basic idea is to hide confidential information in ordinary information. For example, confidential information is embedded and saved in a part of a large file such as an image file. In steganography, there is a risk that the confidential information will be overwritten when the file in which such confidential information is embedded is changed. To avoid overwriting confidential information, the risk is alleviated by copying and saving multiple confidential information, but it has the disadvantage that the usage efficiency of the storage device deteriorates and a large amount of confidential information cannot be retained. PDE using hidden volume technology has been proposed by Jia et al. [9] and Zuck et al. [10]. File system using hidden volume technology creates a decoy volume on a storage device with a decoy key and a hidden volume with a private key. The decoy volume is placed throughout the storage device, and the hidden volume is usually placed from the hidden offset, which is the initial position of the hidden volume on the storage device, toward the end of the storage device. When using the PDE file system using hidden volume technology, the user logs in public mode or PDE mode and uses the file system. In public mode, the user only operates decoy volumes and in PDE mode, the user can operate hidden volumes. When forced to disclose the key, the owner discloses the login password of the public volume and the decoy key, to protect the hidden volume and

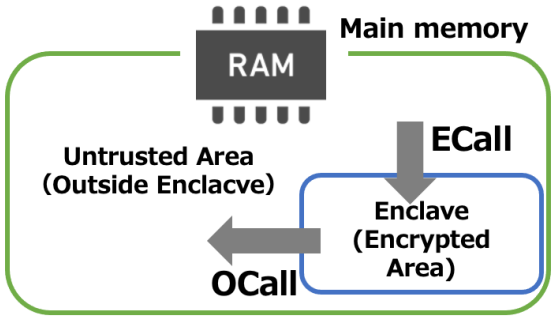

Figure 3: Overview of Intel SGX

the confidential data from the attacker. In the hidden volume technology, the existence of the hidden volume and the hidden offset are unknown in the system that operates the decoy volume, so the data stored in the decoy volume may overwrite the hidden volume.

2.3 Intel Software Guard Extensions

Intel Software Guard Extensions (Intel SGX) [11] is a CPU extension architecture provided by Intel Corporation. Intel SGX can perform processing that guarantees the confidentiality of data even if the privileged user or terminal administrator is not credible. As shown in Fig. 3, Intel SGX creates an encrypted area called Enclave on memory. Enclave provides a trusted execution environment (TEE) to enable program execution while maintaining data confidentiality provided at the hardware level. Intel SGX can protect the programs and data in the enclave from memory inspection attacks. Enclave is called using ECall from untrusted areas. Then, the result processed in the enclave is passed to the untrusted area using OCall. Enclave is executed by the CPU in a special mode which deny cannot be inspected and tampered with by a program outside Enclave. ECall and OCall can achieve confidentiality by denying access from cached addresses to Enclave's private memory by a program outside Enclave. Intel Corporation provides the Intel Software Guard Extensions SDK [12] as an environment for using Intel SGX technologies.

However, Enclave has a limited size that included both program and data, the size is about 100MB. Therefore, the content to be processed by the enclave must be minimized. For example, In the existing research [13] using Intel SGX by Ahemed et al., The policy is to keep only the private key and perform only the related processing in the enclave. A study measuring the performance of Intel SGX by Gjerdrum et al. [14] has shown that the overhead increases when the size of the buffer sent to the enclave exceed 64 kB.

2.4 Measured Traffic for File Server on the Cloud

In the evaluation, we need to assume a usage model of file sharing on the cloud. Leung et al. [6] measured traffic for two file-sharing servers used in NetApp data centers for three months. One of the servers was used by the marketing, sales, and finance departments, and the other was used by the engineering department. This time, we referred to the statistical data of the servers used in each department of marketing, sales, and finance. This server received 364.3GB Read and 177.7GB Write access in 3 months. The ratio of Read, Write, and Delete requests was 540: 170: 1 in this order. The request size when accessing the file was about 70 % for less than 1kB, about 10 % for 1kB or more and less than 100kB, and about 20 % for those over 100kB.

2.5 TCG Storage Security Subsystem Class: Opal

MBR Shadow, one of the functions of "TCG Storage Security Subsystem Class: Opal" [15], allows users to create and switch between the two areas such as decoy and hidden, realized by PDE in the boot process. The area division and access control functions provided by MBR Shadow will solve the issues in overwriting confidential data by decoy data, discussed in Jia et al. [9] and Zuck et al. [10]. However, MBR Shadow configures LBA (Logical Block Addressing) mapping at the time of authentication in the boot process. Therefore, if we configure a storage system with this device to the proposed system, the following usage scenarios required for this system cannot be realized. When a legitimate user and an attacker use the system at the same time, it is required that simultaneous and parallel access needs to be allowed to each of the hidden and decoy areas.

3 PLAUSIBLY DENIABLE DISTRIBUTED FILE SYSTEMS

The purpose of this research is to realize a plausibly deniable distributed file system that is resistant to key disclosure attacks and also resists memory inspection attacks in virtual environments.

3.1 Base Design

In our research so far [5], we have designed a prototype of a distributed file system for key disclosure attacks as follows.

The basic idea of PDE is that using a decoy key or passphrase will give you information that is allowed to be disclosed and using the original private key or passphrase will give you highly confidential information. To realize the basic idea of PDE, the proposed system provides a mechanism to switch the contents of the file handled based on the key and passphrase used for logging in to the file system.

PTEE FS server operates only the encrypted file and does not operate the plaintext file, but the PTEE FS client encrypts and decrypts the data and operates plaintext files. The server manages the decoy space and the hidden space. In the hidden area, highly confidential data such as access keys and passphrases for other systems that should not be leaked are stored. The decoy area does not include the data to be saved in the hidden area, and the data with low risk even if leakage occurs to the outside is saved. PTEE FS sever has the authorization control unit that determines whether the key sent from the client is a decoy or authentic and switches the operation to protect it using TEE (Trusted Execution Environment) and

performs processing. We use the NFS (Network File System) protocol with necessary modifications.

In the proposed configuration, it is necessary to switch the access destination into the decoy area and the hidden area by the key presented by the client and switch the structure of the file system. Code of the structure operation executes in TEE to prevent leakage and inspection by a snapshot of the main memory.

Since Intel SGX is used as the TEE, the confidentiality of the code for these structural operations can be maintained even when the attacker is a privileged user or terminal administrator. Therefore, this configuration can be resistant to infringement from snapshots of the main memory when accessing the file system. However, with the TEE built using Intel SGX, there is a limit to the size of the enclave that can be used, and there are some that cannot be used for kernel functions such as standard input/output in the enclave. In this research, we consider the security of the parts that are not protected by TEE and propose the system configuration that protects them.

It is possible to obtain resistance to infringement from snapshots of persistent storage devices by performing processing such as filling empty areas on the file system with random bits as by Jia et al. [9]. This is because NAND flash devices such as SSD have the characteristic that the entire block is filled with " 1" bit when the block is deleted. Therefore, it is possible to judge whether the area is free or used from the snapshot of persistent storage. If the block is not filled with " 1" bit even though it is a free area on the file system that handles the decoy area, the attacker can suspects the existence of confidential data. Therefore, processing such as filling the free area on the file system with random bits is performed.

3.2 Key Authorization

In this configuration, decryption is performed by the client, so the key or password between server and client send for access authentication and authorization at the time of mounting. A key used for encryption/decryption, a key, and a passphrase used for authorization with the PTEE FS server is not the same. The key for encryption/decryption is handled only on the client terminal. The authorization control unit determines whether the key sent from the client is a decoy or authentic and switches the operation to protect it using TEE and performs processing. The system configuration at this time is as shown in Fig. 4. Figure 4 shows the data flow when the client access the persistent storage device mounted by the Read call. The system that decrypts and encrypts at the client terminal is called PTEE FS Client, and the system that determines the key and switches the operation at the server is called PTEE FS Sever. The user who uses the client terminal receives the encrypted data as shown in Fig. 4 at the client terminal, decrypts it with the PTEE FS Client, and uses it. A legitimate user who uses the proposed system usually uses the private key, and when the attacker forces the decryption key to be disclosed, the decoy key is disclosed to protect the confidential data.

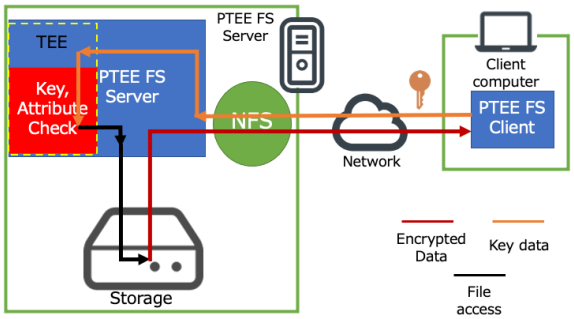

Figure 4: Data flow using TEE in the proposed system

4 PROBLEM

We give a design to resist attacks using knowledge from the disclosure of the decoy key and obtain a practical prospect from the performance estimation when applied to cloud services. In addition to the attack methods we have examined so far, we describe attacks that use knowledge from the disclosure of decoy keys that have not been examined so far.

We introduce the PDE system and threat analysis as assumptions for our previous proposal in Section 4.1. We have provided improvements to these threats in our previous proposals. Then, we explain an attack using knowledge from disclosed decoy key in Section 4.2. We present the design of this system and estimate the performance given by TEE when operating with the access pattern of the file synchronization service that is often seen in cloud storage services. In addition, we evaluate the performance of the system proposed in Chapter 5 when it is used in a typical workload when using cloud storage based on the existing research by Leung et al. [6].

4.1 Threat Analysis

Make some assumptions for threat analysis. First, a legitimate user who accesses the server on which the proposed system is running always mounts the server and handles the data. Next, the attacker has the same access rights to the cloud server as the cloud provider, and the attacker can take snapshots of the main memory and persistent storage at any time. Finally, the attacker does not have access to the legitimate user's client terminal. Based on these assumptions, the vulnerable points of the system configuration will be described. Figure 5 shows the server's basic configuration of the file system which is the conventional method using the PDE concept and the places where the denial of the existence of the hidden area may be lost. In Fig. 5, it is assumed that the persistent storage device is encrypted by the hidden volume and the main memory does not use a protection mechanism such as TEE. In Fig. 5, the location is indicated by a red circle and labeled with the number written on the speech balloon. In the following, in Fig. 5, the part with label 1 is referred to as "vulnerable point 1", and the part with labels 2 and 3 is referred to as "vulnerable point 2" and "vulnerable point 3". The attack model for each vulnerable point is as follows.

vulnerable point 1： Infringement from a running applica-

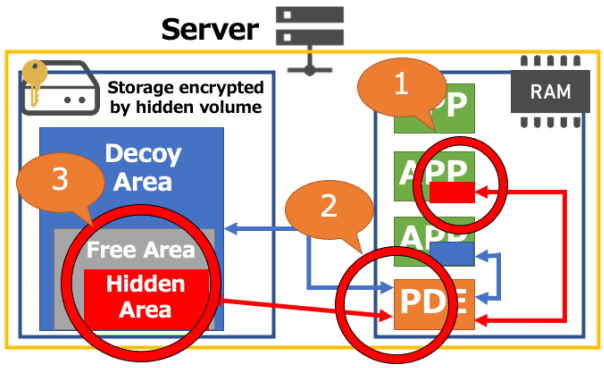

Figure 5: Vulnerable points in the basic configuration of a hidden volume file system

tion for user use

vulnerable point 2： Infringement when accessing the file system

vulnerable point 3： Infringement from persistent storage snapshots

These attacks were given resistance by the base design described in Section 3.1. For infringement from a running application for user use, the proposed system has a server/client configuration., and resistance is given by using the data only on the client terminal that cannot be observed by the attacker. For infringement when accessing the file system, resistance to attacks is given by executing a program related to hidden areas such as code that handles the file structure, on the TEE of the server. For Infringement from persistent storage snapshots, resistance is given by filling the free space with random bits as shown in Section 3.1. However, in the prototype design, there is no discussion about attacks using knowledge from the use of disclosed the decoy key.

4.2 Exploiting Knowledge from the Use of Disclosed Decoy Key

We explain an attack that uses knowledge from the disclosure of the decoy key. When an attacker whose decoy key is disclosed can acquire the time series of attacker's access information to the decoy area by network traffic or a memory inspection attack on the server, the time series of access information to the hidden area by the private key by the legitimate user can be obtained, and the existence of the hidden area is revealed by comparing and collating these.

Regarding attacks using the knowledge of decoy key disclosure in PTEE FS, we will consider how the attacks are possible by monitoring the data exchange at the interface of TEE, and how to protect them. Figure 6 shows the data flow in the TEE interface. There are two interfaces, one between the network and TEE and the other between the persistent storage device and TEE. The information that can be observed in each interface is defined as follows.

TS1: (TimeSeries1) In the operation time series between the network and TEE, the exchange of the modified NFS protocol is observed.

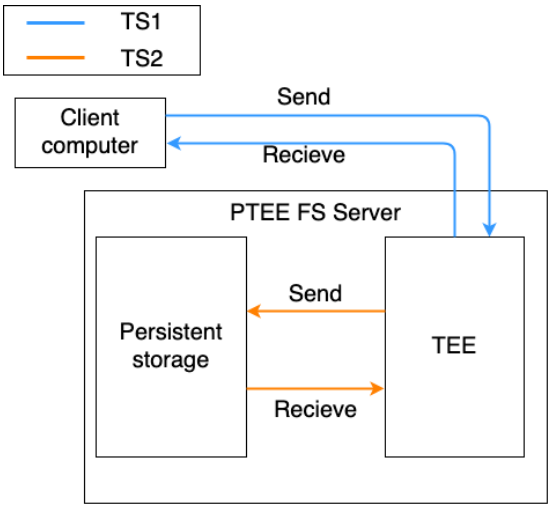

Figure 6: Data flow in TEE interface

Table 1: Example of TS1 and TS2

TS1	Send	Recieve
	getattr File	(OK, Error) Result
	getattr Dir	(OK, Error) Result
	readdirplus Dir	(OK, Error) Result
	write File data	(OK, Error) Result
	read File	(OK, Error) Result data
TS2	Send	Recieve
	fetch ObjectID data	Result ObjectID (OK, Error)
	store ObjectID data	Result ObjectID (OK, Error)

TS2: (TimeSeries2) In the operation time series between the persistent storage device and TEE, operation sequences such as fetch and store to the persistent storage device are observed.

Table 1 below shows examples of the contents observed by TS1 and TS2.

TS1 is represented by the blue line in Fig. 6, and TS2 is represented by the orange line. TS1 and TS2 are arbitrarily generated by an attacker as TS1_m and TS2_m (m: malicious), and those generated by legitimate user operations are TS1_l and TS2_l (l:legitimate). At this time, the following two attacks can be considered from the information observable on the TEE interface.

Attack Possibility 1: Because of the attacker observing the difference between TS1_m and TS1_l, the existence of the hidden area is revealed

Attack Possibility 2: When TS2_m is externally observable as an operation result of TS1_m, it is possible to judge the match between TS1s from the unification of the pair of TS2_m and TS2_l, and the hidden area Existence is exposed

We place two assumptions are made as conditions for establishing "attack possibility 2".

Attacker Assumption 1: Correspondence between TS1 and TS2
$TS2 = TEE\_exposed\_func(TS1)$
can be estimated. This means that it is possible to associate the operation series from the NFS RPC time series with the operations for the persistent storage device.

Attacker Assumption 2: It is possible to judge the match between the elements of TS2. In other words, it means that the operations on the persistent storage device can be identified and the unification can be observed.

Therefore, the following two are required to protect confidential information from attackers using the proposed method.

1. "Attack Possibility 1" is not established
2. Defend "Attack potential 2" by disabling either "Attacker Assumption 1" or "Attacker Assumption 2".

4.2.1 Eliminating Attack Possibility 1

By encrypting the payload part of the RPC of the packet, which is a component of TS1, the difference other than the data size becomes unobservable, and the occurrence of "Attacker Possibility 1" can be prevented.

4.2.2 Eliminating Attack Possibility 2

For "Attack Possibility 2", the following system configuration is adopted to prevent the "Attacker Assumption 1" from being established. As with the countermeasure for "Attack Possibility 1", the part related to RPC of the packet is encrypted. Regarding TS2, the data itself stored in the persistent storage device will be encrypted. In this configuration, the persistent storage device side assumes a general disk or a normal file system, so the object ID used when specifying the target in the persistent storage device is not protected from memory inspection attacks. The information obtained by the attacker at this time is the operation and object ID, the input/output timing to TEE, and the size of the encrypted part. The appearance pattern of the object ID in the IO traffic at the TEE mustn't provide any clue for the attacker tracks hidden volume using TS2.

5 DESIGN OF PTEE FS

FID is the object ID used by the attacker to specify in the persistent storage device. To solve the problem of attacks using the knowledge from the use of disclosed decoy key, the file ID (FID) observed by the attacker in TS2 should be the same in the decoy and hidden area. We propose FID unification to achieve and maintain this. This way, even when accessing sensitive data, only FID known to the attacker in the decoy area is observed. Specifically, the file block specified by one FID has a structure that holds decoy data and confidential data inside. Figure 7 shows the structure of a file block with the same file ID. Place sensitive metadata immediately after decoy metadata, then write decoy data, then sensitive data. Confidential metadata and confidential data are encrypted with the private key, and for users who only know the decoy key,

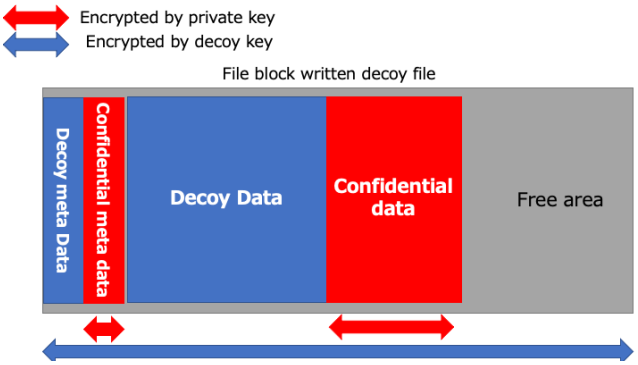

Figure 7: About File block that decoy and hidden FID become same

these data look like a random bit. The state where the FID is the same in the decoy and the hidden area may be destroyed by the file operation by the user. For example, when a decoy file is deleted, a file block that holds only confidential data occurs. After that, by accessing that confidential data, the deleted FID is observed. Section 5.2 defines the procedure for FID unification to maintain the FID unification during operation and to perform FID unification for the entire file system by initialization and so on.

5.1 Discussion about FID Unification

When considering an attack using knowledge from the use of disclosed decoy key, the worst case is when the attacker obtains all the information of the files existing in the decoy area, and the amount of knowledge about the storage area is maximized. This situation is realized when the decoy key is disclosed to only one attacker, there is no access by a legitimate user, and there is no use other than this attacker who changes the decoy area. At this time, the attacker can count and examine all the file IDs existing in the decoy area. After that, the access of the legitimate user is monitored and the file ID that appears based on this knowledge is inspected, and if the attacker observes a file ID that does not exist in the list of counted IDs, the attacker can suspect the existence of a hidden file. Here, consider another strategy, for example, a mechanism in which a common area accessible from a legitimate user and an attacker is created and a decoy file ID and a common file ID appear randomly. Assuming the worst-case above, where there is no legitimate user access and only this attacker to change the decoy area. The attacker can count and examine all the file IDs that exist in the common area and the decoy area in this case. Therefore, the common area is equivalent to the expanded decoy area and is not an effective measure. However, for example, assuming the existence of parallel access accompanied by an update to a file by a legitimate user when observed by an attacker, it seems possible to create a certain confidential margin.

To solve the problems of these methods, propose a method for a file in any hidden area to be embedded internally in one of the files in the decoy area. Here, the hidden file is recognized as a free area by the system that handles only the decoy area. Similarly, one directory in a hidden area should be

embedded inside one of the directories in the decoy area. As a premise, the data placed on the decoy side or hidden side in the persistent storage object read is recorded in the encrypted area of the persistent storage object. Which one should be accessed by the system can be safely confirmed and operated within TEE.

5.2 FID Unification Procedure

The operation of embedding a hidden file inside a file in a decoy area in an appropriate directory structure is called FID unification processing. In the FID unification process, the same FID can be used by embedding the contents of the hidden area file in the file located in the appropriate directory structure of the decoy area. Embedding this file is called a merge operation.

The ideal design of the FID unification process is executed on the server-side, monitoring the existence of non-identical FIDs for each file operation and merging files as necessary. However, if the merge process requires an amount of calculation in the design, the client may be blocked for a long time. To evaluate the amount of the processing, we implemented the FID unification process on the client-side to simplify the design enough for performance estimations. Figure 8 shows a simple client-side flow of the FID identification process. It is assumed that the FID unification process will be called after some extent of updates are performed on the client-side and when the client's PC idle state continues.

The FID unification process and merge operation are shown below. The FID unification process is used for initialization immediately after the proposed system is applied and for the reunification of unmerged files caused by file changes during operation. A simplified flow of the processing in the FID unification process is shown in Fig. 9. To merge the files in the hidden area into the files in the appropriate decoy area, the combination is searched to identify the appropriate location of the directory structure in the decoy area by the method shown in Algorithm1.

Algorithm1 operates as follows. First, get the pathname list of all directories in the decoy area and the hidden area, and pass them to Function Search as an argument. In Function Search, the directory position of the decoy area, which is the starting point of the FID unification process, is determined from the combination of all the directories of the decoy area and the hidden area. The determination method is as follows. From the directory position of the decoy area that is the starting point, each directory of the decoy area and the hidden area has a one-to-one correspondence, and the following unification suitability evaluation is calculated by the operation shown in Alogorithm2. The calculation method of the unification conformity assessment in Algorithm2 is explained in Section 5.2.1. The unification relevance evaluation consists of a mergeable flag and a conformance score. The mergeable flag is expressed by a boolean value indicating whether the directory combination can be merged, and when true, it indicates that the merge condition is satisfied. The calculation of the unification suitability evaluation is made into a memo, and when it is necessary to calculate the score of the same combination, it is called from the memo to shorten the

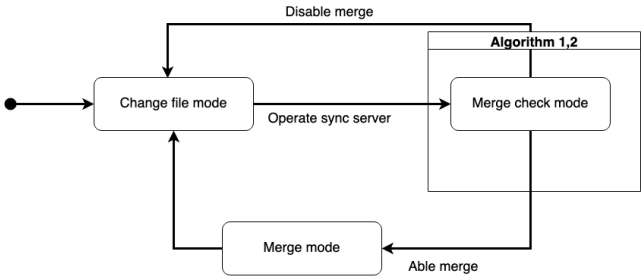

Figure 8: FID unification and operation mode on client PC

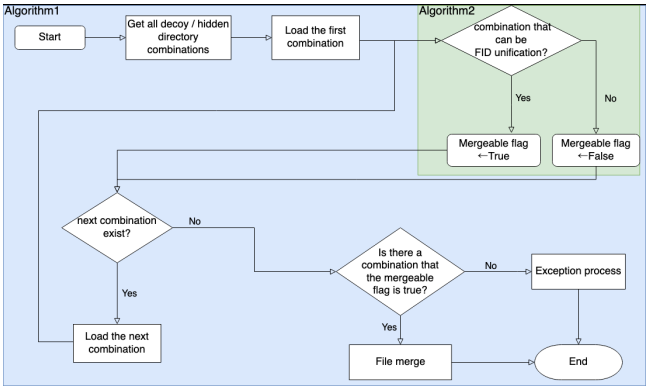

Figure 9: Simplified flow in FID unification

calculation.

Among all combinations, the one with the maximum optimal score is selected from the ones for which the mergeable flag is true, and the directory position of the decoy area that is the starting point of the FID unification process is determined. If none of all combinations have the mergeable flag set to true, the one with the highest matching score is taken out and judged to be at risk based on that combination. Algorithm1 performs the processing up to this point and returns the directory location of the decoy area that is the starting point of the FID unification process, or risk. The FID unification process recursively merges files or notifies the user of the risk based on the result received from Algorithm1. There is a risk, that is, the mergeable flag obtained by Algorithm2 is not true because there are not enough decoy files in the decoy directory to be merged. Therefore, it calculates how many files should be added to which directory in the decoy area, and also notifies the user.

5.2.1 Conformance Score

The conformance score integrates the conformance values for a specific file to be merged, and the larger the conformance score, the better the combination of the corresponding directories. A high match score means that the percentage of files in the decoy area where hidden area files are not embedded is high. In other words, if the conformance score is high, even if a new file on the hidden side is added or a file on the decoy side is deleted, there is a high possibility that the FID unification process can be performed only within the combination of the corresponding directories. It is used as a conformance

score of the FID unification process. The average size of the files in the directory is the size of the decoy area as pSize, and the size of the hidden area is as sSize. The number of files in the directory is the number of decoy areas as pNum, and the number of hidden areas as sNum. The mergeable flag is boolean valule of the following predicate equation.

$$(pSize/sSize + pNum/sNum)/2 >= 2$$

The conformance score is given by;

$$pNum/sNum$$

5.3 Invocation of FID Unification Process

We have two cases in the FID unification process invoked. First, FID unification processing is performed in the initialization of this system. Start by migrating the server to the same mode for FID unification processing. Perform the above FID unification, notify the user of the result if necessary, and then switch to normal mode. Next, the processing after the changes are made to the file will be explained. Think about the directory containing the file that is unmerged in the hidden area due to the modified file. Target hidden directory has already been merged into the decoy area directory. When re-merging a file that has been changed and is out of the merged state, first check whether it can be resolved in the corresponding directory. Algorithm1 is operated by passing only the corresponding directory as an argument. The relevance evaluation of the directories of the corresponding decoy area and the hidden area is calculated, and if the mergeable flag is true, remerge is performed in the corresponding decoy area directory and the hidden area directory. If the mergeable flag is false, the mode shifts to the unification mode in the same way as the initialization, and the FID unification process is started for all directories.

6 EXPERIMENT

In this section, to consider the validity of the design of PTEE FS, the evaluation is performed using the verification case from the following two points.

1. Processing time in normal access :
For the performance when applied to the cloud service of Section 6.1, first, we get the trace data of the file system acquired under assuming a realistic file group workload. The processing time is estimated applied our performance model [5] to the trace data got.

2. Processing time of FID unification invoked on demand
: Regarding the FID unification processing shown in the proposed method, the processing time when applied to the local file system is measured by the evaluation program. The evaluation program is implemented in Algorithm1and Algorithm2, algorithms are implemented in python. Estimate from actual measurement and extrapolation of processing time using the evaluation program.

6.1 Processing Time for Normal Access

For the implementation model of the proposed method, we estimate the performance when operating with the access pattern of the file synchronization service that is often seen in cloud storage services. We obtained Equation (1) as a model for calculating the overhead on the processing time per call to NFS RPC that needs protection [5].

OH (ms) represents the overhead of additional response time, and i (MB) represents the size of the transfer buffer during the enclave call. o (MB) represents the size of the transfer buffer when escaping the enclave.

Since it is known that the overhead increases when the transfer buffer at the time of enclave call exceeds 64 kB, the equation is divided into 2 by $i = 64kB$ when calculating the overhead. When the proposed system is applied to a cloud service, it can be seen that the effect on the performance of the file system can be evaluated from the difference in response time depending on whether TEE is used or not.

File synchronization is a practical example of cloud applications. Therefore, we will evaluate TEE performance again and use RSYNC [16] as a concrete file synchronization system. Trace the RSYNC execution traffic, give the size to be sent/received as an argument of Equation (1), calculate the estimation overhead, and evaluate its effectiveness.

$$OH = \begin{cases} 0.0097 + 0.0354o + 0.0115 & i < 0.064 \\ 0.9560i + 0.0354o - 0.0002 & i >= 0.064 \end{cases} \quad (1)$$

6.2 Processing Time of FID Unification Process

Regarding the FID unification process, we evaluate the effect of the smallest process among the FID unification processes that occur when used in a typical workload. When the same usage as the file system using the existing PDE concept is used, it is the most called process in the proposed method, and the impact on the user is significant. So, we evaluate the effect of the additional latency to gave by the FID unification process.

Create an environment that assumes the use case described in Section 7 of the proposed system, operate the FID unification process under that environment, and perform an evaluation experiment.

The performance is estimated and evaluated from the response time by the Python-based code for the effect of the FID unification process on the performance in a steady state. In this experiment, we evaluate a part of the performance of FID unification processing. In the experiment, we perform the process of passing only one directory to each argument of Alogrithm1, which occurs most in the FID unification process.

The FID unification process may be performed on all directories existing in the volume, but this is not the target of this experiment. As a PTEE FS system, it is conceivable that decryption and encryption will be performed before and after the FID unification process, but this is not the case in this experiment. The performance of the implementation in Python may be lower or almost unchanged than that implemented in

Algorithm 1 Algorithm to search for the best directory combination

```
function Serach(secretDirs, publicDirs)
    if secretDirs.length > publicDirs.length then ▷ If the hidden area has more directories than decoy area, no search is performed because there is no matching pattern.
        result ⇐ noMatch
        return result
    allMatch ⇐ allPermutaitionPatern(publicDirs) ▷ Calculate and substitute permutation patterns for directories in all decoy areas
    for i = 1, ··· , allMatch.length do ▷ Repeat the process for the number of allMatch
        for j = 1, secretFileNum do ▷ Repeat the process for the number of file in hidden area
            if resultMemo[j][allMatch[i][j]] = null then
                score ⇐ CheckMatchDir(secretDirs[j], publicDirs[allMatch[i][j]]) ▷ Get the mergeable flag and optimal value for a combination of a directory in a decoy area and a directory in a hidden area
                resultMemo[j][allMatch[i][j]] ⇐ score ▷ Save the score you have done once in a memo
            else
                score ⇐ resultMemo[j][allMatch[i][j]] ▷ When the same combination appears, call it from the memo
            throghScore[i].optimal ⇐ score.optimal ▷ Accumulate scores in the current permutation pattern
            if score.conform = false then
                throghScore[i].conform ⇐ false
                throghScore[i].optimal ⇐ −1
    if max(throughScore[i].optimal)! = 1 then ▷ Check if there is a mergeable combination
        result ⇐ argmax(throughScore.optimal) ▷ Get the permutation pattern with the highest conformance score
    else
        result ⇐ noMatch
    return result
```

C ++. Therefore, if a sufficient value is obtained in the evaluation performance based on Python, it is expected that the evaluation performance will be sufficient in C ++. The proposed system will be implemented in C ++, but this time we will evaluate it using the one implemented in Python.

To understand the additional latency provided by the FID unification process, compare the processing time of the FID unification process in one directory in the local directory to which PTEE FS has already been applied and the processing time of file synchronization with PTEE FS. The following use cases are assumed when applying the FID unification process.

It is assumed that the client uses this system when saving highly confidential data such as password lists and keys in the storage on the cloud for backup purposes. For this reason, the data saved in the hidden area is sufficiently smaller than the decoy area, and even if the FID unification process occurs due to file changes/deletion, it is assumed that most of the data can be resolved only in the relevant directory.

The assumed decoy use case workload is as follows. The ratio of requests for create: read: write is set to about 1: 40.5: 12.75, referring to the previous research by Leung et al. [6]. The size ratio of bytes flowing on the traffic by read: write is 2.1: 1. 70% of the file size to be handled is a file less than 1KB, 10% is 1 to 100KB, and the remaining 20% is 100KB to 1MB. The hidden area use case assumes pgp key management, and the key used is 2048 bits of RSA encryption key. The processing time of the FID unification process is evaluated as the additional latency that gives the proposed system as it is, and the effect of the additional latency on the file system is estimated and evaluated concerning an example of file sharing.

6.3 Experimental Method

6.3.1 Experiment of Processing Time with Normal Access

We prepared a server/client capable of RSYNC communication and synchronized the files. The server configuration of RSYNC is mainly done by directly using the file server in the company or school, but it is assumed that the changes are synchronized on the client terminal at home for remote work. Assuming that the size of the data loaded for uploading and downloading the modified file is the same, here we trace the traffic between the uploading client and server. Figure 10 shows the configuration for data acquisition. The server was mounted on the proxy server, rsync was performed from the client to the proxy server, and NFS traffic between the proxy server and server was observed. From the trace data, the RPC READ and WIRTE call get the series of buffer sizes to be passed.

The series of buffer sizes was used as the buffer size for calling and escaping the enclave in Equation (1), the execution time was measured, and the evaluation was performed as the overhead of Intel SGX when using this system.

The response time of the entire process was measured 10 times for file synchronization of the same size, and the average process time was used. The files shown in Table 2 were used as the files that are changed synchronously by RSYNC. The average size of the text file was 11263 bytes, and the average size of the binary file was 216269 bytes. Four of the files have a file size that exceeds 64 kB, which increases the TEE response time to input/output these files.

Algorithm 2 Algorithm for calculating the unification aptitude score

```
1: function CheckMatchDir(secretDir, publicDir)
2:     publicFiles ⇐ getAllFiles(publicDir)          ▷ Get the file entry for the target decoy area directory
3:     secretFiles ⇐ getAllFiles(secretDir)          ▷ Get the file entry for the target hidden area directory
4:     publicFileSizeMean ⇐ publicFiles.sumSize/publicFiles.fileNum   ▷ Get the average file size of the decoy area directory
5:     secretFileSizeMean ⇐ secretFiles.sumSize/secretFiles.fileNum   ▷ Get the average file size of the hidden area directory
6:     if publicFileSizeMean/secretFileSizeMean > 1 then   ▷ Check if the average file size meets the conditions
7:         sizeScore ⇐ true
8:     else
9:         sizeScore ⇐ false
10:    if publicFiles.fileNum > secretFiles.fileNum then   ▷ Check if number of files meets the conditions
11:        fileNumScore ⇐ true
12:    else
13:        fileNumScore ⇐ false
14:    if sizeScore&fileNumScore then
15:        conform ⇐ true
16:    else
17:        conform ⇐ false
18:    if conform then   ▷ Check if both the average file size and number of files meets the conditions
19:        optimal ⇐ publicFiles.fileNum/secretFiles.fileNum   ▷ If the mergeable flag is true, the optimum value is calculated.
20:    else
21:        optimal ⇐ 0
22:    return (conform, optimal)   ▷ Returns mergeable flag, conformance score
```

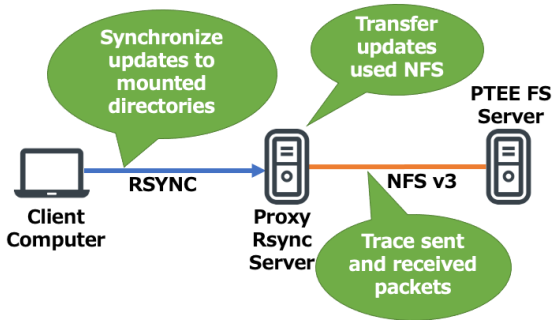

Figure 10: Server configuration in the experiment

Table 2: File size and type used in the experiment

File size (byte)	File type	File size (byte)	File type
55358	Text	1444438	Binary
22113	Text	223065	Binary
11602	Text	97252	Binary
11101	Text	96516	Binary
4284	Text	35184	Binary
2651	Text	33033	Binary
2607	Text	10244	Binary
2317	Text	6527	Binary
567	Text	165	Binary
28	Text		

6.3.2 Experiment of Processing Time of FID Unification Processing

We prepared a decoy area directory and a hidden area directory according to the workload of the use case and performed the FID unification processing. For the decoy area directory, referring to the existing research [6] by Leung et al., We prepared 70% for files with file sizes from 1 byte to 1 kB, 10% for files with a file size of 1 kB to 100 kB, and 20% for files with a file size of 100 kB or more. We prepared three types of files, 30 and 50, contained in one decoy directory. For the hidden directory, referring to the key management of pgp, it was decided that the public key and private key pair of public key authentication, which is asymmetric authentication, is assigned to each directory. Two types of files, 10 and 20, are prepared in one hidden directory. If the number of files is 10, there are 5 public/private key pairs, and if the number of files is 20, there are 10 public/private key pairs. In the actual experiment, assuming that the user uses so that the number of files on the hidden area side is sufficiently small in the PDE file system. We experiment with 2 pairs of decoy area directories and hidden area directories. One pair is that the number of files in the decoy area directory is 30 and the number of hidden area directories is 10. The other is that number of files in the decoy area directory was 50 and the number of hidden area directories was 20. We execute the FID unification processing in these 2 pairs and the execution time was measured.

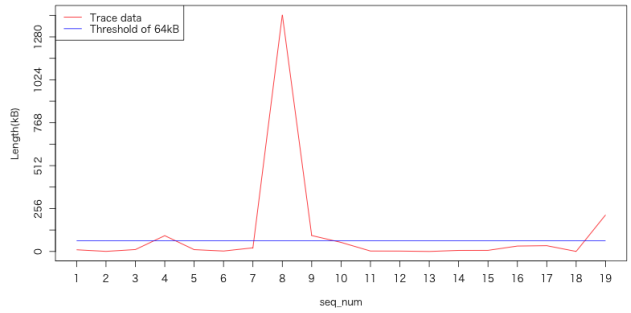

Figure 11: Transfer data size and part of the NFS Call series observed during rsync

6.4 Experiment Environment

A computer with RSYNC [16] and NFS Version 3 [17] installed was used as the server and client for the experiment. Wire Shark [18] was used to trace the traffic. The network bandwidth in the experimental environment was 6.90 MB/s.

7 EVALUATION

7.1 Experiment of Processing Time with Normal Access

The average response time of the entire process on the client/server in the experimental environment was 18336.2 ms, and the standard deviation was 2373. Figure 11 shows a series of transfer data sizes that appear in the NFS traffic trace when RSYNC is executed. [5]. Substituting this transfer data size sequence into Equation (1) for calculating the increase in response time, which is the overhead (OH) per NFS call, gives 1.9 ms from the total sequence. Therefore, the estimated processing time when using the proposed system is 18338.1 ms. Therefore, the ratio of overhead to the processing time is 0.010%, and it is considered that the overhead required when using TEE is acceptable.

7.2 Experiment of Processing Time of FID Unification Processing

The average time required for FID unification is 133.8 ms with 30 files in the decoy area and 10 files in the hidden area, and 134.6 ms with 50 files in the decoy area and 20 files in the hidden area.

8 CONCLUSION

We improved our design of Plausibly Deniable Distributed File Systems to obtain resistance to key disclosure attacks. Two experiments were conducted and evaluated in terms of performance to validate the design. In the experiments and evaluations, we discussed the processing time for normal access in use cases applied to cloud services. In the file synchronization use case using rsync, the increased ratio in response time by the use of TEE is estimated with a measured figure. The result is 0.010%. Increase for the whole operation, which is considered to be acceptable overhead by TEE. To provide the resistance to exploiting the knowledge from the use of disclosed the decoy key, we added new functionalities of the FID unification as a countermeasure to memory inspection attacks. The processing time of the FID unification process invoked on-demand was tested and evaluated using a program implemented in python.

The processing time of the FID unification process is 1133.8 ms in an environment with 30 files in the decoy area and 10 files in the hidden area, and 134.6 ms with 50 files in the decoy area and 20 files in the hidden area. Therefore, the additional latency due to the FID unification process may be tolerated. However, in this evaluation, the cost of encryption processing is not added to the processing time. Examination of a performance model that includes these is future work.

REFERENCES

[1] R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky. “Deniable Encryption”. In B. S. Kaliski, editor, Advances in Cryptology — CRYPTO ’97, Lecture Notes in Computer Science, pp. 90–104. Springer Berlin Heidelberg, (1997).

[2] Ministry of Internal Affairs and Communications. “Ministry of Internal Affairs and Communications | 2019 WHITE PAPER Information and Communications in Japan | ICT Data on the ICT Field”. https://www.soumu.go.jp/johotsusintokei/whitepaper/eng/WP2019/chapter-3.pdf#page=1. (accessed 2019-10-22).

[3] G. Kumparak. “Capital One hacked, over 100 million customers affected”. https://social.techcrunch.com/2019/07/29/capital-one-hacked-over-100-million-customers-affected/. (accessed 2021-06-11).

[4] H. Ryo, T. Sasaki, Y. Morita, K. Miyoshi, and T. Kobayashi. “A Study on Access Control for Devices with Trusted Execution Environments”. Proceedings of Computer Security Symposium 2017, Vol. 2017, No. 2, (2017/10/16).

[5] R. Shibazaki, H. Inamura, and Y. Nakamura. “Design of Encrypted File System Using the Concept of PDE”. Proceedings of the 82th National Convention of IPSJ, Vol. 82, No. 1, pp. 103–104, (2020).

[6] A. W. Leung, S. Pasupathy, G. Goodson, and E. L. Miller. “Measurement and Analysis of Large-Scale Network File System Workloads”. In 2008 USENIX Annual Technical Conference (USENIX ATC 08), (2008/6).

[7] R. Anderson, R. Needham, and A. Shamir. “The Steganographic File System”. In Information Hiding, pp. 73–82. Springer, Berlin, Heidelberg, (1998/4/14).

[8] B. Chang, F. Zhang, B. Chen, Y. Li, W. Zhu, Y. Tian, Z. Wang, and A. Ching. “MobiCeal: Towards Secure and Practical Plausibly Deniable Encryption on Mobile Devices”. In 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 454–465, (June 2018).

[9] S. Jia, L. Xia, B. Chen, and P. Liu. “DEFTL: Implementing Plausibly Deniable Encryption in Flash Translation Layer”. In Proceedings of the 2017 ACM SIGSAC

Conference on Computer and Communications Security, CCS '17, pp. 2217–2229, New York, NY, USA, (2017). ACM.

[10] A. Zuck, U. Shriki, D. E. Porter, and D. Tsafrir. "Preserving Hidden Data with an Ever-Changing Disk". In Proceedings of the 16th Workshop on Hot Topics in Operating Systems, HotOS '17, pp. 50–55, New York, NY, USA, (2017). ACM.

[11] "Intel® Software Guard Extensions (Intel® SGX)". https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions.html. (accessed 2021-06-11).

[12] Intel® Software Developer Zone. "SDK — Intel® Software Guard Extensions". https://software.intel.com/en-us/sgx/sdk. (accessed 2019-10-26).

[13] R. Ahmed, Z. Zaheer, R. Li, and R. Ricci. "Harpocrates: Giving Out Your Secrets and Keeping Them Too". In 2018 IEEE/ACM Symposium on Edge Computing (SEC), pp. 103–114, Seattle, WA, USA, (10/2018). IEEE.

[14] A. Gjerdrum, R. Pettersen, H. D. Johansen, and D. Johansen. "Performance of Trusted Computing in Cloud Infrastructures with Intel SGX:". In Proceedings of the 7th International Conference on Cloud Computing and Services Science, pp. 696–703, Porto, Portugal, (2017/4). SCITEPRESS - Science and Technology Publications.

[15] Trusted Computing Group. "TCG Storage Security Subsystem Class: Opal", (2022/1/24). (accessed 2022-02-16).

[16] rsync. "rsync". https://rsync.samba.org/. (accessed 2020-05-08).

[17] B. Callaghan, B. Pawlowski, and P. Staubach. "NFS Version 3 Protocol Specification". https://www.ietf.org/rfc/rfc1813.txt, (1995 June). (accessed 2019-12-24).

[18] WIRESHARK. "Wireshark". https://www.wireshark.org. (accessed 2020-02-13).

(Received: October 26, 2021)
(Accepted: April 26, 2022)

Ryouga Shibazaki received B.E. and M.S. degree in from Future University Hakodate in 2020 and 2022. His research interests include cloud file system and OS security. He is currently an engineer in MEITEC CORPORATION.

Hiroshi Inamura He is a professor of School of Systems Information Science, Future University Hakodate, since 2016. His current research interests include mobile computing, system software for smart devices, IoT network and their security. He was an executive research engineer in NTT docomo, Inc. He received B.E., M.E. and D.E. degree in Keio University, Japan. He is a member of IPSJ, IEICE, ACM and IEEE.

Yoshitaka Nakamura received B.E., M.S., and Ph.D. degrees from Osaka University in 2002, 2004 and 2007, respectively. He is currently an associate professor at the Faculty of Engineering , Kyoto Tachibana University. His research interest includes information security and ubiquitous computing. He is a member of IEEE, IEICE, and IPSJ.