A Design of Plausibly Deniable Distributted 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 disclosuse 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 exploit the knowledge from the use of disclosed 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 are 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 storage, 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.

So far, the purpose of this study is to construct an encrypted file system that is resistant to attacks not only on the permanent storage device but also on the main storage device and that can deny the existence using the concept of PDE. We proposed a system using Intel SGX as a hardware-protected execution environment in the realization of an encrypted file system[2].

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). 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 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[3] 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, has the problem that it 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 Figure 1, 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 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 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, idea of PDE that decoy key gives decoy information and private key gives the confidential information can be used.

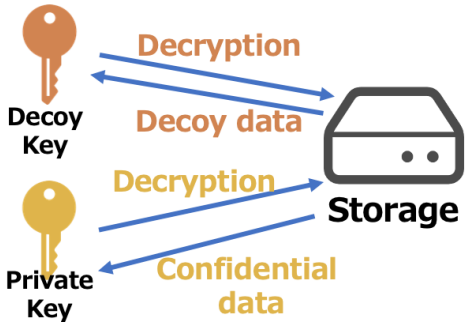

Figure 1: Overview of PDE

2.2 Applications of PDE concept

Using 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.[4] and Chang et al.[5]. 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. In order to avoid overwriting confidential information, 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.[6] and Zuck et al.[7]. File sytem 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 PDE file system using hidden volume technology, the user logs in in public mode or PDE mode and uses the file system. In public mode, user only operate decoy volumes and in PDE mode, user can operate hidden volumes. When forced to disclose the key, the owner discloses the login password of public volume and the decoy key, so that protect hidden volume and 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)[8] 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 Figure 2, Intel SGX create an encrypted area called Enclave on memory. Enclave provide 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 are 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 by program outside Enclave. ECall and OCall can achieve confidentiality by deny access from cached address to Enclave's private memory by program outside Enclave. Intel Corporation provides the Intel Software Guard Extensions SDK as an environment for using Intel SGX technologies.

However, Enclave has a limit 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[9] 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.[10] has shown that the overhead increases when the size of the buffer sent to the enclave exceeds 64 kB.

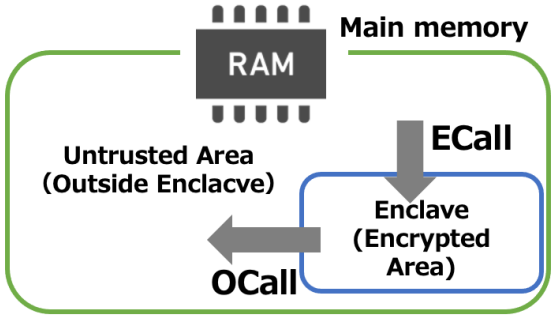

Figure 2: Overview of Intel SGX

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. [3] 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.

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 resist memory inspection attacks in virtual environments.

3.1 Base Design

In our research so far[2], 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. In order 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 PTEE FS client encrypts and decrypts the data and operate plaintext files. The server manages the decoy space and the hidden space. In the hidden area, highly confidential data such as access keys and passphrase 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 decoy or authentic and switches the operation protects it using TEE (Trusted Execution Environment) and performs processing. The legitimate client PC and TEE are reliable areas, and the keys and passphrases used for user authorization are handled only in those areas. We use the NFS (Network File System) protocol with necessary modifications.

In propose 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 execute 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 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.[6].

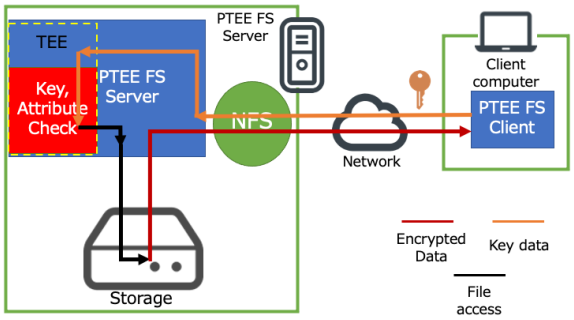

Figure 3: Data flow using TEE in the proposed system

4 Problem

We give a design resist attacks used knowledge from disclosure of 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. Next, we explain 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.[3].

4.1 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 4 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.

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.

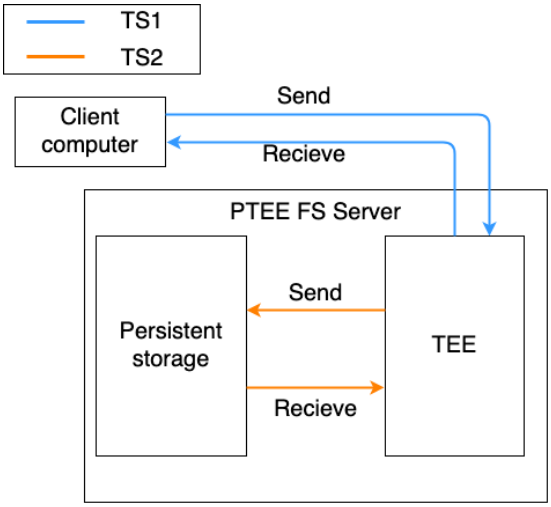

Figure 4: Data flow in TEE interface

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

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)

TS1 is represented by the blue line in Fig. 1, and TS2 is represented by the orange line. TS1 and TS2 are arbitrarily generated by an attacker as TS1_m and TS2_m (m: malicous), 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 to the operations for permanent 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".

4.1.1 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.

4.1.2 Eliminating Attack Possibility 2

For "Attack Possibility 2", the following system configuration is adopted in order 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. It is necessary that the appearance pattern of the object ID in the IO traffic at the TEE does not provide any clue for attacker's tracking using TS2.

5 Design of PTEE FS

To solve the problems mentioned in Section 1, the object ID used by the attacker to specify in the persistent storage device observed by the TS2 should be the same between the decoy area and the hidden area as much as possible. To achieve this, there is a method of files that exist in any hidden area is 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, where the decoy side or hidden side data exists in the persistent storage object actually read is recorded in the encrypted area of the persistent storage object. It is safely confirmes or operated in TEE which one should be accessed now.

5.1 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 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 re-unification of unmerged files caused by file changes during operation. In order 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 path name list of all directories in the decoy area and the hidden area, and pass them to Funciton Search as an argument. In Funciton 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 5.1.1. 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 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 that 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.

5.1.1 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, 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 calculation of the mergeable flag is $(pSize/sSize + pNum/sNum)/2 >= 2$ The calculation of the conformance score is $pNum/sNum$

6 Experiment

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

Processing time in normal access :

For the performance when applied to the cloud service of Section **??**, 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 [2] to the trace data got.

Regarding the FID unification process, we evaluate the effect of the smallest process among the FID unification processes that occurs 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 FID unification process.

Create an environment that assumes the use case described in Section 6 of the proposed system, operate the FID unification process under that environment, and perform an evaluation experiment.

6.1 Experimental method

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[3] by Leung et al., We prepared 70% for files with file sizes from 1 byte to 1 kB, 10%

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.
    $result \Leftarrow noMatch$
    return $result$
  $allMatch \Leftarrow allPermutaitionPatern(publicDirs)$ ▷ Calculate and substitute permutation patterns for directories in all decoy areas
  for $i = 1, \cdots, allMatch.length$ **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**
        $score \Leftarrow CheckMatchDir(secretDirs[j], publicDirs[allMatch[i][j]])$ ▷ Get the mergeable flag and optimal value for a combination of a directory in a decoy area and a directory in a hidden area
        $resultMemo[j][allMatch[i][j]] \Leftarrow score$ ▷ Save the score you have done once in a memo
      else
        $score \Leftarrow resultMemo[j][allMatch[i][j]]$ ▷ When the same combination appears, call it from the memo
      $throghScore[i].optimal \Leftarrow score.optimal$ ▷ Accumulate scores in the current permutation pattern
      if $score.conform = false$ **then**
        $throghScore[i].conform \Leftarrow false$
        $throghScore[i].optimal \Leftarrow -1$
  if $max(throughScore[i].optimal)! = 1$ **then** ▷ Check if there is a mergeable combination
    $result \Leftarrow argmax(throughScore.optimal)$ ▷ Get the permutation pattern with the highest conformance score
  else
    $result \Leftarrow noMatch$
  return $result$

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 user use so that the number of files on the hidden area side is sufficiently small in PDE file system. We experiment 2 pairs of decoy area directory 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 excute the FID unification processing in these 2 pairs and the execution time was measured.

6.2 Experiment environment

A computer with RSYNC and NFS Version 3[11] installed was used as the server and client for the experiment. Wire Shark was used to trace the traffic. The network bandwidth in the experimental environment was 6.90 MB/s.

7 Evaluation

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 resistant to key disclosure attacks. Two experiments were conducted and evaluated in terms of performance in order 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 radio in response time by the use of TEE is estimated with measured figure. The result is 0.010%. increase for whole operation, which is considered to be acceptable overhead by TEE. To provide the resistance to exploit the knowledge from the use of disclosed 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 a future work.

Algorithm 2 Algorithm for calculating the unification aptitude score

function $CheckMatchDir(secretDir, publicDir)$
  $publicFiles \Leftarrow getAllFiles(publicDir)$ ▷ Get the file entry for the target decoy area directory
  $secretFiles \Leftarrow getAllFiles(secretDir)$ ▷ Get the file entry for the target hidden area directory
  $publicFileSizeMean \Leftarrow publicFiles.sumSize/publicFiles.fileNum$ ▷ Get the average file size of the decoy area directory
  $secretFileSizeMean \Leftarrow secretFiles.sumSize/secretFiles.fileNum$ ▷ Get the average file size of the hidden area directory
  if $publicFileSizeMean/secretFileSizeMean > 1$ **then** ▷ Check if the average file size meets the conditions
    $sizeScore \Leftarrow true$
  else
    $sizeScore \Leftarrow false$
  if $publicFiles.fileNum > secretFiles.fileNum$ **then** ▷ Check if number of files meets the conditions
    $fileNumScore \Leftarrow true$
  else
    $fileNumScore \Leftarrow false$
  if $sizeScore\&fileNumScore$ **then**
    $conform \Leftarrow true$
  else
    $conform \Leftarrow false$
  if $conform$ **then** ▷ Check if both the average file size and number of files meets the conditions
    $optimal \Leftarrow publicFiles.fileNum/secretFiles.fileNum$ ▷ If the mergeable flag is true, the optimum value is calculated.
  else
    $optimal \Leftarrow 0$
  return $(conform, optimal)$ ▷ Returns mergeable flag, conformance score

REFERENCES

[1] Rein Canetti, Cynthia Dwork, Moni Naor, and Rafail Ostrovsky. "Deniable Encryption". In Burton S. Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, Lecture Notes in Computer Science, pp. 90–104. Springer Berlin Heidelberg, 1997.

[2] Shibazaki Ryouga, Inamura Hiroshi, and Nakamura Yoshitaka. "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.

[3] Andrew W. Leung, Shankar Pasupathy, Garth Goodson, and Ethan L. Miller. "Measurement and Analysis of Large-Scale Network File System Workloads". In *2008 {USENIX} Annual Technical Conference ({USENIX} {ATC} 08)*, 2008.

[4] Ross Anderson, Roger Needham, and Adi Shamir. "The Steganographic File System". In *Information Hiding*, pp. 73–82. Springer, Berlin, Heidelberg, April 1998.

[5] 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.

[6] Shijie Jia, Luning Xia, Bo Chen, and Peng 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.

[7] Aviad Zuck, Udi Shriki, Donald E. Porter, and Dan 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.

[8] "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).

[9] Rufaida Ahmed, Zirak Zaheer, Richard Li, and Robert 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, October 2018. IEEE.

[10] Anders T. Gjerdrum, Robert Pettersen, Håvard D. Johansen, and Dag 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. SCITEPRESS - Science and Technology Publications.

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