TerraCheck: Verification of Dedicated Cloud Storage

Zhan Wang[1,2], Kun Sun[2], Sushil Jajodia[2], and Jiwu Jing[1]

[1] State Key Laboratory of Information Security
Institute of Information Engineering
Chinese Academy of Sciences, Beijing 100093, China
{zwang, jing}@lois.cn
[2] Center for Secure Information Systems
George Mason University, Fairfax, VA 22030, USA
{ksun3, jajodia}@gmu.edu

Abstract.

When hardware resources are shared between mutually distrustful tenants in the cloud, it may cause information leakage and bring difficulties to regulatory control. To address these concerns, cloud providers are starting to offer hardware resources dedicated to a single user. Cloud users have to pay more for such dedicated tenancy; however, they may not be able to detect the unexpected misuse of their dedicated storage due to the abstraction layer of the cloud. In this paper, we propose *TerraCheck* to help cloud users verify if their dedicated storage devices have been misused to store other users' data. *TerraCheck* detects the malicious occupation of the dedicated device by monitoring the change of the shadow data that are residual bits intentionally left on the disk and are invisible by the file system. When the cloud providers share the dedicated disk with other users, such misuses can be detected since the shadow data will be overwritten and become irretrievable. We describe the theoretical framework of *TerraCheck* and show experimentally that *TerraCheck* works well in practice.

Keywords: Dedicated Storage, Cloud Security, Verification

1 Introduction

Cloud service significantly reduces costs by multiplexing hardware resources among users [12]. The co-resident data belonging to different users may lead to information leakage, which has become a major security concern for cloud users. For instance, a malicious VM is capable of retrieving the encryption keys [21] from a victim VM hosted on the same physical machine. Sensitive information can be compromised through the covert communication channels based on the shared CPU cache [19], memory bus [18], hard disks [15, 17] and so on in the cloud.

Cloud providers [1] are starting to offer physically isolated resources in order to lower the entry barrier of cloud adoption for processing sensitive data. For instance, in Amazon cloud [1], *Dedicated Instances* are a form of EC2 instances launched within the Amazon Virtual Private Cloud (Amazon VPC) that runs on hardware dedicated to a single customer. A dedicated instance ensures that the resources, such as CPU, memory, disk storage and network, are isolated physically at the hardware level. Unsurprisingly, the cloud users have to pay more for the dedicated resources than the regular ones.

Although the dedication property is guaranteed by the Service Level Agreement (SLA), a misbehaved cloud provider may fail to meet the isolation requirement due to either accidental configuration error or intentionally reassigning the unallocated resources to other users. As a consequence, the dedicated resource, for example the storage device, will store the data belonging to unexpected users and cause information leakage. Because the cloud users usually can only see a logical view of their resources due to the abstraction layer or the business model of cloud computing [11], they may not be aware of or not be able to detect the violation of the desired dedicated configuration before the security breaches occur.

In this paper, we propose *TerraCheck* to help cloud users to verify if the unallocated disk space has been occupied by undesired users without the cooperation of the cloud provider. We assume that the cloud providers are *honest-but-greedy*, i.e., trustworthy for managing user's data without violating the data privacy but *greedy* for allocating the storage resources not being in use by the dedicated user to other tenants. To detect the greedy allocation, *TerraCheck* places shadow data on the unallocated disk space and verifies the dedication by detecting the change of the shadow information.

The shadow data are the residual bits on the disk after deleting the original stored data and invisible to the file system. We group the set of residual bits related to the same original file as "shadow chunk". We record the hash value and physical disk address of each shadow chunk as verification metadata. To verify the integrity of each shadow chunk, we utilize disk forensics techniques to retrieve shadow chunks according to the prior recorded disk addresses. If the shadow chunks cannot be recovered entirely, it indicates that the unallocated disk space has been overwritten and the dedication property is violated.

Our shadow chunk method has two advantages comparing to simply stuffing the unallocated disk space with void files. First, it makes the cheating behavior of the *honest-but-greedy* cloud provider very costly.

The retrieval of the shadow chunks relies on the physical disk address of each chunk. If the misbehaved cloud providers move the shadow data to some non-dedicated devices and make the shadow data still retrievable, they must map the prior recorded disk address to the addresses of the new device. Instead, accessing files relies on the file system and can be redirected to another device with less efforts. Second, shadow data will not affect the normal use of the dedicated device. The attested disk area filled by the shadow chunks remains available for allocation in the view of file system. However, if the attested disk area is filled by files, it cannot be occupied by the dedicated user immediately.

We present two schemes for verifying the dedication property of cloud storage. The basic *TerraCheck* scheme can detect the unexpected occupation of dedicated storage device with high accuracy by checking the retrievability of every chunk. With sampling, our advanced probabilistic *TerraCheck* scheme can discover 10% unexpected occupation of the dedicated storage device with 95% probability by randomly challenging 29 chunks. Therefore, smaller size of chunk achieves low computational cost but results in the large storage of metadata. Furthermore, with the help of Bloom filter with 1% false positive rate, the size of verification metadata can be reduced 5.5 times.

The rest of the paper is organized as follows. In Section 2, we describe the threat model and general assumptions. Section 3 presents the requirements and operations of the dedication verification. Section 4 describes both the basic and advanced probabilistic *TerraCheck* schemes. Section 5 implements two schemes and evaluates both the computational and storage costs. Section 6 overviews the related work. Section 7 concludes this paper.

2 Threat Model and Assumptions

The dedication property of cloud storage is guaranteed by the terms in SLA. However, a misbehaved cloud provider may fail to meet such dedication requirement due to either accidental configuration errors or intentionally being greedy with the unallocated storage resources: First, configuration error may allocate dedicated storage space to undesired tenants. For instance, in Amazon dedicated instance, the dedication property is enabled by the "Dedicated" attribute configured at the launch time. The "Dedicated" attribute may be silently disabled (e.g., for software update, server migration or testing). Second, a cloud provider may intentionally

place the non-frequently accessed data, such as archive data, to the unoccupied disk space where is supposed belonging to one specific customer.

We consider the misbehaved cloud providers as *honest-but-greedy*. *Honest* means that the cloud providers are not motivated to corrupt user's data or violate the data privacy with respect to the business reputation. However, the cloud providers may be *greedy* for allocating the storage not being in use by the dedicated user to other tenants. Although the *honest-but-greedy* cloud providers are only interested in the large amount of unused disk space belonging to a dedicated user, they cannot control the behavior of the co-resident tenants once the cloud provider accidentally allocate the unoccupied space to another tenant. Co-resident tenants may threaten the security and privacy of the existing user data, such as exploiting covert channels to retrieve encryption key [21] and other sensitive information [15] or violating the access control policy [17].

We assume the usage of the dedicated storage is well-planned by the user. For example, the user allocates a determined amount of dedicated disk space to each VM. This is a common practice [11] of resource management in the cloud. When the user only launches a small number of VMs, only part of the dedicated storage are allocated. The rest of the dedicated storage should be protected from being exploited by other users due to both the security and performance reasons. We call this part of disk space as *attested area*. The disk space being in use by the dedicated user is called *occupied area*. Additionally, the attested area may scale up and down based on the occupation of the dedicated disk. *TerraCheck* requires a small amount of trusted disk space for storing verification metadata on the *occupied area*. We assume the *occupied area* is trusted since *honest-but-greedy* cloud provider is trustworthy for managing user data.

3 System Model

We first formalize the model of *TerraCheck*. Suppose a user C pays and possesses a dedicated disk with the capacity of s in the cloud. The dedicated disk is divided into two areas as shown in Fig. 1. The *occupied area* with the capacity of s_a disk space has been allocated by C for storing the data associated with running VMs or as general purpose storage. We consider occupied area is trusted by C to execute the *TerraCheck* and store the verification metadata. The *attested area* with the capacity of s_u disk space remains unallocated where $s_u = s - s_a$. Attested area is the verification target of *TerraCheck*. When C needs more disk space by increasing the size of occupied area, the size of attested area will shrink

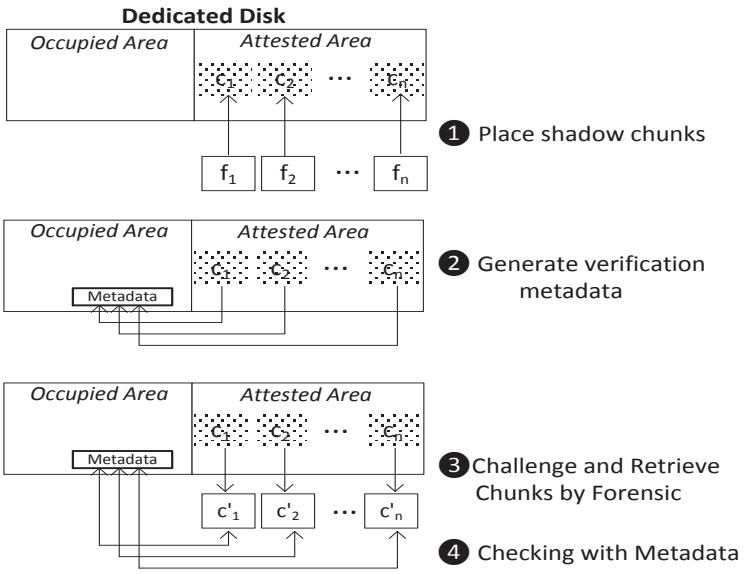

Fig. 1. Overview of *TerraCheck*

accordingly. The goal of *TerraCheck* is to verify if the attested area has been maliciously taken by other users or the cloud provider.

TerraCheck consists of four major procedures, as shown in Fig. 1. First, it places shadow chunks on the attested area of the target disk. The shadow chunks are deleted files which cannot be accessed from the file system. Shadow chunks can be recovered by disk forensics technique as long as they have not been overwritten. Second, it generates metadata, such as the hash value of the shadow chunks, for monitoring the alternation of shadow chunks. The metadata are stored on the occupied area where has been allocated for storing the data associated with running VMs or as general purpose storage. Third, *TerraCheck* challenges the shadow chunks by using disk forensic technique to recover them. Lastly, it compares the forensics results with the verification metadata. If any one of the shadow chunks has been altered and cannot be recovered, a violation of dedication property is detected.

3.1 Verification Requirements

A solution for verifying the dedicated storage should satisfy the following technical requirements.

– *Trustworthy.* The verification mechanism should provide the users high confidence on the result of the verification. In other words, the

Table 1. Summary of Operation Parameters

Variable	Meaning
C	The cloud user who possesses the dedicated device and executes dedication verification
n	The number of shadow chunks placed on attested disk area
l_k	Length of each shadow chunk
t_h	Header tag of each chunk
t_f	Footer tag of each chunk
K	The set of shadow chunks
s_u	Size of unallocated disk space
id_{k_i}	ID of shadow chunk i
F	The set of files for generating shadow chunks
img_{AA}	Disk image of attested area
$meta_{DB}$	File for storing verification metadata
b_i	Starting disk address of chunk i on attested area
e_i	Ending disk address of chunk i on attested area
id_{AR_x}	ID of attested region x
$meta_{FILTER}$	File for storing Bloom filter

cloud provider has to pay higher storage overhead to defeat our verification mechanism. We can ensure our checking capability from the economic consideration.

- *Efficiency.* The verification procedure should be fast without obviously interrupting the disk activities against the allocated part of the disk. Moreover, The metadata used for verification should be small; otherwise, it is unacceptable to use the same amount or more of disk space to store the original shadow data on the local disk.
- *Scalability.* When the dedicated user occupies or releases more disk space, for example, for running more VMs or shutting down existing VMs, the disk area to be attested varies. Every time the customer needs to scale the disk space up or down, the affected shadow chunks should be as few as possible.

3.2 System Operations

TerraCheck consists of five basic operations. $ChunkGen$ generates the shadow chunks and places them on the *attested area*. $MetaGen$ generates the verification metadata and stores them on the *occupied area*. $ChalGen$ generates the information of challenged chunks. *Retrieve* executes the forensics of challenged chunks and calculates their hash values. $Verify$ operation compares the result of *Retrieve* with the verification metadata recorded in $MetaGen$ and makes the decision of the dedication verification. Table 1 summarizes all the variables used in this paper.

- ***ChunkGen***$(n,\ l_k,\ t_h,\ t_f) \rightarrow K = \{k_1, k_2, ..., k_n\}$**:** *TerraCheck* fills *attested area* with a set of chunks $K = \{k_1, k_2, ..., k_n\}$ and $n * l_k = s_u$.

Each chunk k_i has a header tag t_h and a footer tag t_f to represent the start and the end of a chunk, respectively. The total length of the header and the footer $l_{t_h} + l_{t_f}$ is less than l_k. This algorithm takes the number of chunks, the length of each chunk, the header t_h, the footer t_f as inputs and generates n temporary files $F = \{f_1, f_2, ..., f_n\}$ first. Every file f_i in F starts with t_h, ends with t_f and the rest of it is filled by random bits. Every file f_i has the same length as l_k. All the files in F are stored on attested area and then deleted from the file system. The bits left on attested area associated with each file f_i are the set of chunks $K = \{k_1, k_2, ..., k_n\}$. Each chunk contains three parts - the header, the footer, and a random body.

- ***MetaGen*(n, t_h, t_f, img_{AA}, h)$\rightarrow${$meta_{DB}$, $\perp$}:** It takes the number of chunks, the header, footer tag information, the disk image of *attested area* and a hash function as inputs, returns the verification metadata or abortion. $h : \{0,1\}^* \rightarrow \{0,1\}^m$ denotes a fixed hash function that outputs m bits hash value. The MetaGen algorithm retrieves the chunks from img_{AA} by matching the t_h and t_f and calculates the hash value of each chunk. The results of verification metadata $meta_{DB}$ is stored on occupied area. $meta_{DB} = \{(id_{k_i}, b_i, e_i, h(k_i)) | i \in \{1, 2...n\}, k_i \in K\}$ lists the ID of a chunk and the boundary of each chunk on the disk, such as the start block number b_i and the end block number e_i of chunk k_i, and the hash value of each chunk $h(k_i)$. Each chunk can be retrieved from the raw disk based on the start and end block number without the help of the file system. Let $|meta_{DB}|$ be the number of items in $meta_{DB}$. If $|meta_{DB}| \neq n$, it indicates that some chunks either cannot be recovered from the disk image of attested area or a mismatched header or footer involved among the chunks. In this case, MetaGen fails and outputs abortion symbol $\perp$.
- ***ChalGen*($meta_{DB}$, id_{k_i})$\rightarrow$*chal*:** This algorithm generates a challenge *chal* based on $meta_{DB}$ and the ID of the queried chunk. $chal = (id_{k_i}, b_i, e_i, h(k_i)) \in meta_{DB}$ is the chunk to be examined.
- ***Retrieve*(*chal*, h)$\rightarrow$*result*:** It takes the challenge and the hash function as inputs and calculates the hash value after retrieving the chunk based on the information specified in *chal*. It returns the hash value of the chunk in *chal*.
- ***Verify*(*result*, *chal*)$\rightarrow${"success", "failure"}:** The Verify algorithm takes *result* and *chal* as inputs and compares the hash value in *result* with that in *chal*. If the two hash values match, it outputs "success" and otherwise outputs "failure".

4 *TerraCheck* Schemes

We propose two schemes. The basic *TerraCheck* can accurately verify the violation of the dedication with a high computation and storage overhead. The advanced *TerraCheck* can detect the violation of the dedication with a high probability while reducing the verification overhead dramatically.

4.1 Basic Scheme

Our goal is to make sure that the attested area hasn't been allocated to other users. Our basic *TerraCheck* scheme consists of four phases.

- **Initial.** In the initial phase, the attested area is filled by all zeros. This operation prevents the existing content on the disk from affecting our placement results.
- **Placement.** We place the shadow chunks on the attested area by using the $ChunckGen$ and $MetaGen$ algorithms. If $MetaGen \rightarrow \perp$, a failure occurs, *TerraCheck* should be restarted from the initial phase. Otherwise, $MetaGen$ generates valid verification metadata $meta_{DB}$.
- **Verification** is a procedure to patrol on the dedicated storage device and collect the evidence for the undesired occupation by calling $Challenge$, $Retrieve$ and $Verify$ algorithms until each shadow chunk placed in the attested area has been checked. The $Verification$ phase would be stopped once $Verify$ algorithm returns a "failure" for any chunk. The dedication property is preserved if all the chunks passed the examination.
- **Update** is executed when the size of attested area is subject to change. It is difficult to predict the set of affected chunks since the allocation of disk space depends on the disk scheduling. Therefore, both the shadow chunks and their associated verification metadata become useless and subjects to deletion. The initial phase and placement phase should be restarted with the new attested area.

The basic *TerraCheck* can successfully check the dedication property with high accuracy. If $n-t$ shadow chunks are recoverable, it means that t chunks are altered so that around $t * l_k$ out of s_u disk space has been allocated or corrupted maliciously. Theoretically, we can 100% detect the alternation of any number of chunks. However, the basic *TerraCheck* scheme has two main limitations:

- **Computational Cost.** The verification phase has to read through the whole *attested area* and calculate the hash value for every shadow chunk.

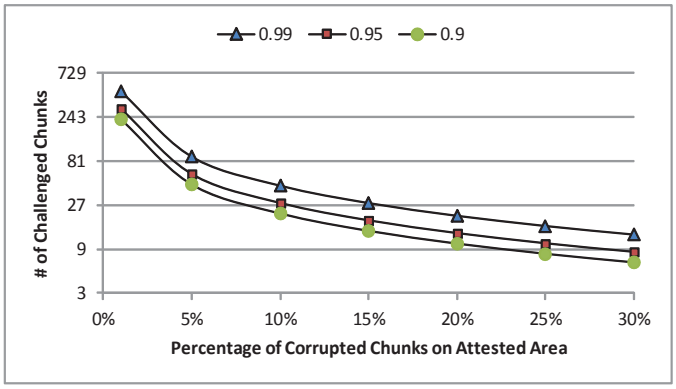

Fig. 2. Probabilistic Framework of Advanced *TerraCheck*

– **Update Operation.** When the size of *attested area* has to be changed, *TerraCheck* should be restarted from the initial phase against the new *attested area*.

4.2 Advanced Scheme

To mitigate the limitations of the basic *TerraCheck* scheme, we propose a probabilistic based *TerraCheck* scheme. To reduce the computational cost, we randomly sample the chunks during the *Verification* procedure. In order to provide efficient update operation, we introduce multiple regions within the attested area, we call them *attested region*. The attested region is the smallest unit for C to scale up the size of the occupied area. For example, C plans to attach a certain size of disk space to a newly launched VM. When the size of occupied area is shrunk due to the termination of a VM, new attested region will be created. Each attested region contains multiple shadow chunks. The shadow chunk is the smallest unit for challenge and verification. In addition, we use bloom filter to reduce the storage for saving the verification metadata.

Attested Region We introduce attested region for conveniently scaling up and down the size of attested area. The attested area is divided into multiple attested regions. The size of attested region depends on how a user uses the dedicated disk. For example, if it uses the disk as the attached secondary storage for running VMs, and each VM is attached by a fixed amount of disk space, such amount is an optimal size for each attested region. When an attested region should be deleted, the related

verification metadata are deleted and excluded from the *TerraCheck* procedure.

Probabilistic Verification The sampling would greatly reduce the computational cost, while still achieving a high detection probability. We now analyze the probabilistic guarantees offered by a scheme that supports chunk sampling.

Suppose the client probes p chunks during the *Challenge* phase. Clearly, if the cloud provider destroys with a chunk other than those probed, the cloud provider will not be caught. Assume now that t chunks are tampered and become unrecoverable, so that at least $s_t = t * l_k$ size of disk space are maliciously allocated. If the total number of chunks is n, the probability that at least one of the probed chunks matches at least one of the tampered chunks is $\rho = 1 - \frac{n-t}{n} \cdot \frac{n-t-1}{n-1}, ..., \cdot \frac{n-p+1-t}{n-p+1}$. Since $\frac{n-t-i}{n-i} \geq \frac{n-t-i-1}{n-i-1}$, it follows that $\rho \geq 1 - (\frac{n-t}{n})^p$.

When t is a fraction of the chunks, user C can detect misbehaviors by asking for a constant amount of chunks, independently on the total number of file blocks. As shown in Fig. 2, if $t = 1\%$ of n, then *TerraCheck* asks for 459 chunks, 300 chunks and 230 chunks in order to achieve the probability of at least 99%, 95% and 90%, respectively. When the number of corrupted chunks goes up to 10% of the total chunks, the violation of dedicated can be detected with 95% probability by only challenging 29 chunks. As the number of corrupted chunks increases, the number of chunks required to be checked is decreased dramatically. The sampling is overwhelmingly better than scanning all chunks in the basic *TerraCheck* scheme. Therefore, we can challenge a fix number of chunks to achieve certain accuracy. The size of each chunk will determine the computation cost. When the size of each chunk is small, the overhead for retrieving all challenged chunks from dedicated disk is low.

Advanced Operations For establishing efficient *TerraCheck*, we need to refine both the $MetaGen$ and $ChalGen$ algorithms.

***MetaGen*(n, t_h, t_f, img_{AA}, h)→{$meta_{DB}$, $\perp$}:** The results of verification metadata $meta_{DB} = \{(id_{AR_x}, id_{k_i}, b_i, e_i, h(k_i)) | i \in \{1, 2...n\}, k_i \in K\}$. It lists the ID of the located attested region, the ID of a chunk and the boundary of each chunk on the disk, such as the start block number b_i and the end block number e_i of chunk k_i, and the hash value of each chunk $h(k_i)$. Each chunk can be retrieved from the raw disk based on the start and end block number and the ID of the attested region without the help of the file system.

ChalGen$(meta_{DB}) \xrightarrow{r} chal$. It randomly generates a challenge *chal* based on $meta_{DB}$. $chal = (id_{AR_r}, id_{k_r}, b_r, e_r, h(k_r)) \in meta_{DB}$ is the chunk to be examined.

Our advanced *TerraCheck* scheme consists of the same phases as the basic *TerraCheck*. Advanced operations will be called in the related phased. Besides, the update phase should be modified accordlingly.

- ***Update.*** Since the attested area is further divided into attested regions, when a user needs to extend or shrink the disk space for occupied area, only limited number of attested regions are deleted or added so that the *TerraCheck* against the rest of chunks remains valid. When the occupied area scales up, the metadata related to the erased attested region will be deleted. The rest of metadata are still available for *TerraCheck*.

Reducing Metadata Storage In the basic *TerraCheck* scheme, the size of $meta_{DB}$ for storing the verification metadata is linear to the number of shadow chunks. The number of chunks could be very large if the user wants to achieve a lower computational cost as we discussed in the probabilistic verification. In order to reduce the amount of storage for verification metadata in *TerraCheck*, we take advantage of Bloom filter to store the metadata for verification.

Bloom filter [4] is a space-efficient data structure for representing a set in order to support membership queries. Bloom filter is suitable to the place where one might like to keep or send a list for verification, but a complete list requires too much space. We use Bloom filter to represent a set $S = \{x_1, x_2, ..., x_n\}$ of n elements as an array of m counters, initially all set to 0. It uses k independent hash functions $h_1, h_2, ..., h_k$ with range [1, m]. For mathematical convenience, we make the natural assumption that these hash functions map each item in the universe to a random number over the range $\{1, ..., m\}$. For each element $x \in S$, the bits $h_i(x)$ are set 1 for $1 \leq i \leq k$. A location can be set as 1 multiple times. To check if an item y is a member of S, we check whether all $h_i(y)$ are 1. If not, then clearly y is not a member of S. If all $h_i(y)$ are 1, we assume that y is in S. We know that a Bloom filter may yield a false positive, where it suggests that an element x is in S even though it is not.

The probability of a false positive for an element not in the set, or the false positive rate, can be estimated, given our assumption that hash functions are perfectly random. After all the elements of S are hashed into the Bloom filter, the probability that a specific bit is still 0 is $PR_{zero} =$

$1 - \frac{1}{m}^{kn} \approx e^{-\frac{kn}{m}}$. The probability of a false positive is $(1 - PR_{zero})^k$. A Bloom filter with an optimal value for the number of hash functions can improve storage efficiency.

We modify our *TerraCheck* model for utilizing Bloom filter to reduce the storage cost of the verification metadata.

- ***BF-MetaGen***$(t_h, t_f, img_{AA}, h) \rightarrow \{meta_{FILTER}, \perp\}$ The algorithm takes the header, footer tag information, the disk image of *attested* area and a hash function as inputs, returns the verification metadata or an abortion. $meta_{FILTER}$ is a Bloom filter which involves the hash value of every shadow chunk.
- ***BF-Verify***$(result, meta_{FILTER}) \rightarrow \{$"success", "failure"$\}$: It takes $result$ and $meta_{FILTER}$ as inputs and checks if the hash value in $result$ is valid and associates with any chunks. If the hash value can be found from $meta_{FILTER}$, the algorithm outputs "success" and otherwise "failure".

5 Implementation and Evaluation

We implement and evaluate both basic *TerraCheck* scheme and advanced *TerraCheck* scheme. All experiments are conducted on a Dell PowerEdge460 server with Intel Core i5 CPU running at 3.10GHz, and with 4096 MB of RAM. The system runs Ubuntu 12.04 (LST) that is configured with Xen Hypervisor. The dedicated storage device is a WestDigital SATA 7200 rpm hard disk with 1TB capacity and 64MB cache. For evaluation purpose, we used SHA-1 as the hash function h. The random values used for challenging the chunks in the advanced *TerraCheck* are generated using the function proposed by Shoup [7]. All data represent the mean of 20 trials.

We implement a large attested area in basic *TerraCheck* and implement an attested region in advanced *TerraCheck* as a logical volume. The occupied area may involve multiple logical volumes. LVM (Logical Volume Management) technology is exploited to automate the update operation when the size of occupied disk space varies. We rely on the retrievability of the shadow chunks on each logical volume to check the dedication property. We utilize Scalpel [14], which is an open source file recovery utility with an emphasis on speed and memory efficiency, to retrieve the shadow chunks based on their header tag and footer tag. To perform file recovery, Scalpel makes two sequential passes over each disk image. The first pass reads the entire disk image and searches for the headers, footers and a database of the locations of these headers is maintained. The

Table 2. Time for Retrieving Chunks

Chunk Size	512KB	1MB	2MB	4MB	8MB	16MB
Retrieve Time	13 ms	15 ms	20 ms	29 ms	48 ms	86 ms

second pass retrieves the files from the disk image based on the location information of the header and footer. Scalpel is file system-independent and will carve files from FATx, NTFS, ext2 and ext3, or raw partitions.

We evaluate both the computation overhead and storage cost during each phase of *TerraCheck*.

Initial Phase. During the initial phase, the attested area is filled by all zeros. The time for this phase is determined by and linear to the size of attested area s_u. It takes about 10 seconds for cleaning 1 GB of the attested area. Both basic *TerraCheck* and advanced *TerraCheck* have the same performance at this phase.

Placement Phase. There are two steps for placing the chunks. The first step is to generate and store the chunks to the attested area. The cost of this operation is determined by the chunk size and the size of attested area. On our testbed, it takes 12 seconds to store 100MB of shadow chunks. The second step is to generate the metadata. It takes 8.198 seconds for Scalpel to scan 1 GB of the attested area in the first pass and store the location information.

Verification Phase. The basic *TerraCheck* examines all the chunks based on the verification metadata recorded in $meta_{DB}$. Therefore, the time for generating the challenge can be ignored. The advanced *TerraCheck* randomly challenges the chunks. The generation of random number takes less than 0.1 ms. The challenged chunks are retrieved from the *attested area* based on the start and end location recorded as the verification metadata. Therefore, the performance is determined by the disk access time. Tab. 2 shows the disk access time in our experiment.

After retrieving the challenged chunks, *TerraCheck* compares the hash value of the retrieved chunk with the verification information. In basic *TerraCheck*, it checks all the chunks residing on the attested area so that the checking time is the time for calculating the hash value of all the chunks. The advanced *TerraCheck* scheme randomly challenges the chunks to achieve the detection of undesired disk occupation. We simulate the behaviors that a proportion of attested area is altered. For instance, if a randomly 1% of an attested area with 10000 chunks are altered, such situation could be detected with a 90% probability by challenging 217 chunks on average, which is closed to the theoretical result.

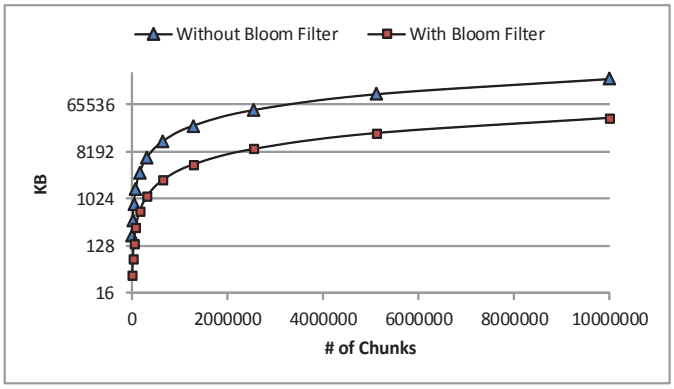

Fig. 3. Comparison of the Storage Cost with/without Bloom Filter (%1 Fault Positive Rate Allowed)

Update Phase. For the basic *TerraCheck* scheme, the performance of update is the same as the overhead of executing the initial and placement phases. The performance of the advanced *TerraCheck* scheme depends on the change of the size of attested area. When the occupied area is extended, the advanced *TerraCheck* scheme only needs to update the $meta_{DB}$ by deleting the items of affected chunks. When occupied area is shrunk, more attested regions should be created on the attested area. The generation of each attested region takes about 400 ms regardless the size of the attested region.

Reducing Metadata Storage. *apgbmf* [2] is originally used to manage Bloom filter for restricting password generation in APG password generation software [16]. We use *apgbmf* version 2.2.3 as a standalone bloom filter management tool.

We consider each hash value of the shadow chunk as an item of password dictionary in the context of *apgbmf*. We create a Bloom filter for such hash value dictionary. During the verification phase of *TerraCheck*, if a recovered chunk is unaltered, its hash value will pass the Bloom filter, i.e, the hash value is one of the hash values which associates an original shadow chunk with a high probability. When we allow 1% fault positive rate, the storage cost with Bloom filter is reduced 5.5 times as shown in Fig. 3. When the number of chunks is more than 10 million, the metadata only requires 36 MB as compared to 200 MB without using Bloom filter.

6 Related Work

Cloud service providers [1, 13] have started to offer physically isolated resources to lower the entry barrier for enterprises to adopt cloud computing

and storage. For instance, in Amazon cloud [1], *Dedicated Instances* are a form of EC2 instances launched within the Amazon Virtual Private Cloud, which runs hardware dedicated to a single customer. Some research have been done to guarantee the exclusive occupation of dedicated resources for security reason. The side channel based on CPU L2 cache has been used to verify the exclusive use of a physical machine [20]. Ristenpart et al. [15] also propose to use the existing side channels to verify the co-residency of VMs. [10] allows application designers to build secure applications in the same way as on a dedicated closed platform by using a trusted virtual machine monitor. However, it requires the modification of commercial hypervisor.

Researchers have also investigated techniques to verify various security properties claimed in the SLAs. Dijk et al. [8] prove that the files are stored with encryption at the cloud server side by imposing a resource requirement on the process of translating files from the plain texts to the cipher texts. Proof of Retrievability (PoR) [9] aims to verify if the files are available in the cloud storage at any time. However, PoR cannot verify where the files are located. RAFT [5] can verify that a file is stored with sufficient redundancy by measuring the response time for accessing "well-collected" file blocks. Another work [3] proposes a mechanism to verify that the cloud storage provider replicates the data in multiple geolocations by measuring the network latency. [17] proposes a method to verify the disk storage isolation of conflict-of-interest files so that Chinese Wall security policy [6] can be successfully enforced in cloud storage environment.

7 Conclusion

In this paper, we propose *TerraCheck* to help cloud users to verify the exclusive use of their dedicated cloud storage resources. *TerraCheck* places shadow chunks on the dedicated disk and detects the change of the shadow information by taking advantage of disk forensics technique. We further improve the computational efficiency by randomly challenging the chunks and reduce the storage by applying Bloom filter.

References

1. Amazon Web Services: `http://aws.amazon.com`
2. APGBFM: `http://linux.die.net/man/1/apgbfm`
3. Benson, K., Dowsley, R., Shacham, H.: Do you know where your cloud files are? In: CCSW. pp. 73–82 (2011)

4. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM 13(7), 422–426 (1970)
5. Bowers, K.D., van Dijk, M., Juels, A., Oprea, A., Rivest, R.L.: How to tell if your cloud files are vulnerable to drive crashes. In: ACM Conference on Computer and Communications Security. pp. 501–514 (2011)
6. Brewer, D.F.C., Nash, M.J.: The chinese wall security policy. In: IEEE Symposium on Security and Privacy. pp. 206–214 (1989)
7. Dent, A.W.: The Cramer-Shoup encryption scheme is plaintext aware in the standard model. In: EUROCRYPT. pp. 289–307 (2006)
8. Dijk, M.V., Juels, A., Oprea, A., Rivest, R.L., Stefanov, E., Triandopoulos, N.: Hourglass schemes: How to prove that cloud files are encrypted. In: ACM Conference on Computer and Communications Security (2012)
9. Dodis, Y., Vadhan, S.P., Wichs, D.: Proofs of retrievability via hardness amplification. In: Theory of Cryptography Conference. pp. 109–127 (2009)
10. Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M., Boneh, D.: Terra: a virtual machine-based platform for trusted computing. SIGOPS Oper. Syst. Rev. 37(5), 193–206 (Oct 2003)
11. Jhawar, R., Piuri, V.: Fault tolerance management in iaas clouds. In: Proc. of the 1st IEEE-AESS Conference in Europe about Space and Satellite Telecommunications (ESTEL 2012). ESTEL 2012, Rome, Italy (Oct 2012)
12. Kurmus, A., Gupta, M., Pletka, R., Cachin, C., Haas, R.: A comparison of secure multi-tenancy architectures for filesystem storage clouds. In: Middleware. pp. 471–490 (2011)
13. Rackspace: `http://www.rackspace.com`
14. Richard III, G.G., Roussev, V.: Scalpel: A frugal, high performance file carver. In: DFRWS (2005)
15. Ristenpart, T., Tromer, E., Shacham, H., Savage, S.: Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In: ACM Conference on Computer and Communications Security. pp. 199–212 (2009)
16. Spafford, E.: Opus: Preventing weak password choices
17. Wang, Z., Sun, K., Jajodia, S., Jing, J.: Disk storage isolation and verification in cloud. In: Globecom 2012. Anaheim, CA, USA (2012)
18. Wu, Z., Xu, Z., Wang, H.: Whispers in the hyper-space: High-speed covert channel attacks in the cloud. In: the 21st USENIX Security Symposium (Security'12) (August 2012)
19. Xu, Y., Bailey, M., Jahanian, F., Joshi, K.R., Hiltunen, M.A., Schlichting, R.D.: An exploration of L2 cache covert channels in virtualized environments. In: CCSW. pp. 29–40 (2011)
20. Zhang, Y., Juels, A., Oprea, A., Reiter, M.K.: Homealone: Co-residency detection in the cloud via side-channel analysis. In: IEEE Symposium on Security and Privacy. pp. 313–328 (2011)
21. Zhang, Y., Juels, A., Reiter, M.K., Ristenpart, T.: Cross-VM side channels and their use to extract private keys. In: Proceedings of the 2012 ACM conference on Computer and communications security. pp. 305–316. CCS (2012)