Reconciling Security and Functional Requirements in Multi-tenant Clouds

Ghassan O. Karame* Matthias Neugschwandtner◇ Melek Önen‡
Hubert Ritzdorf†
* NEC Laboratories Europe, Heidelberg, Germany.
◇ IBM Research Zurich, Rüschlikon, Switzerland.
‡ EURECOM, Sophia-Antipolis, France.
† ETH Zurich, Zurich, Switzerland.

ABSTRACT

End-to-end security in the cloud has gained even more importance after the outbreak of data breaches and massive surveillance programs around the globe last year. While the community features a number of cloud-based security mechanisms, existing solutions either provide security at the expense of the economy of scale and cost effectiveness of the cloud (i.e., at the expense of resource sharing and deduplication techniques), or they meet the latter objectives at the expense of security (e.g., the customer is required to fully trust the provider).

In this paper, we shed light on this problem, and we analyze the challenges in reconciling security and functional requirements in existing multi-tenant clouds. We also explore the solution space to effectively enhance the current security offerings of existing cloud storage services. As far as we are aware, this is the first contribution which comprehensively explores possible avenues for reconciling the current cloud trends with end-to-end security requirements.

Keywords: Cloud security, multi-tenancy, secure deduplication, resource isolation, shared ownership.

1. INTRODUCTION

With the ever increasing amount of data produced worldwide, the cloud offers a cheaper and a more reliable alternative to local storage. Existing cloud service providers such as Amazon S3 and Microsoft Azure, or backup service providers like Dropbox and Memopal guarantee a good trade-off between quality of service and cost effectiveness. Cost effectiveness in the cloud is typically achieved through the extensive use of multi-tenancy solutions combined with efficient distributed algorithms that run on top of simple hardware, which ensure high levels of scalability and elasticity. The combination of multi-tenancy solutions with storage efficiency techniques (e.g., compression, deduplication) promises drastic cost reductions and high service availability.

The advent of cloud storage and computation services, however, comes at the expense of data security and user privacy. Indeed, most cloud providers nowadays deploy standard security solutions by which they retain full control over the customers' data in order to ensure that their offerings can leverage the multitude of benefits originating from the adoption of multi-tenancy and storage efficiency techniques. This entails retaining cryptographic keying material, choosing the underlying cryptographic primitives, etc. This strategy does not only increase the profitability of the cloud, but also ensures that cloud providers can offer cheap services to users at relatively small costs. Unfortunately, in this model, customers of cloud services have no means to control and verify, how data is processed or stored.

Ideally, cloud customers would like to gain back full control over their services without having to depend on the best-effort mechanisms deployed by cloud providers. This is especially true after the outbreak of the PRISM revelations. These revelations unearthed the details of a massive surveillance program which was not restricted to one geographical area, nor was it mitigated by the various security countermeasures already deployed within the targeted services. As a consequence, companies and individuals which make use of clouds are less keen to rely on standard cloud security solutions and call for end-to-end security whereby only end-users and authorized parties can have access to their data/services and no-one else.

However, achieving end-to-end security in existing cloud services is not straightforward. Notably, end-to-end security aims at ensuring that end-users are the only entities able to decrypt their encrypted data outsourced to the cloud. This implies that cloud service providers may not be able to offer standard APIs to efficiently process customers' data, nor may they be able to take full advantage of cost-effective storage solutions which rely on existing deduplication and compression mechanisms. In addition to data confidentiality, users also call for resource isolation solutions that enable tenants to have a secure and "isolated" environment. Although implementing resource isolation in the cloud would indeed deter threats originating from unknown vulnerabilities and the subsequent loss of data governance, these measures typically come at odds with multi-tenancy and resource sharing—which would in turn prevent a successful and cost-effective instantiation of the current multi-tenant cloud model.

Existing state of the art solutions completely give up one requirement for the other. That is, they either rely on standard solutions at the expense of end-to-end security and governance, or provide end-to-end security but renounce any form of resource sharing or storage efficiency techniques. This explains the reason for which, none of today's cloud storage services provide security guarantees in their Service Level Agreements (SLAs) [1], in spite of the plethora of cloud security solutions that populate the literature.

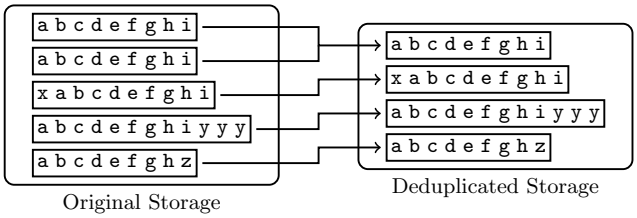

Figure 1: Example of deduplication. Here, exact file copies are only stored once.

While customers' needs are associated with an end-to-end secure and isolated platform, cloud service providers always look for means to minimize their expenditures either by compressing the data or by sharing the resources among multiple tenants. Table 1 highlights the main tension between security and functional requirements in the cloud.

In this paper, we shed light on each challenge depicted in this table and explore the solution space for reconciling security and functional requirements in the cloud. More specifically, we outline the shortcomings of existing data confidentiality and multi-tenancy resource isolation solutions in the literature for supporting the functional requirements of today's required cost-effective clouds. We then sketch a number of solutions for enhancing security and data governance in the cloud without compromising storage efficiency and resource sharing.

We summarize our contributions as follows:

- We analyze the tension between data confidentiality, and storage efficiency solutions that are currently adopted in modern multi-tenant clouds and sketch a number of potential solutions addressing this problem.
- We review existing techniques for isolating resources in multi-tenant cloud environments and outline their limitations in enabling resource sharing among various tenants.
- We explore the tradeoffs between resource isolation and shared ownership in the cloud, and we outline a solution that supports shared ownership in existing cloud platforms (which enforce isolation).

The remainder of this paper is organized as follows. In Section 2, we review current secure deduplication technologies and analyze their application to the cloud model. In Section 3, we discuss and analyze the tension between resource isolation and resource sharing in multi-tenant clouds. In Section 4, we explore the tradeoffs between resource isolation and shared ownership in the cloud. Finally, we conclude the paper in Section 5.

2. CONFIDENTIALITY VS. STORAGE EFFICIENCY AND MULTI-TENANCY

Multi-tenant cloud service providers always seek means to save their storage costs. One way to maximize their storage space is to perform *cross-user deduplication* which consists of detecting duplicate data segments among several customers' data and storing only one copy of them together with the information about whom they belong to (cf. Figure 1). Recent studies show that cross-user data deduplication can save storage costs by more than 50% in standard file systems [11], and by up to 90-95% for back-up applications [11]. Such techniques are already adopted by large providers, such as Dropbox, IBM, EMC, Microsoft, Oracle, etc. Notice that additional savings can be achieved through *client-side deduplication* which effectively occurs before the actual upload of their data. Here, customers check the existence of this data on the cloud server (by sending a hash of the data) and send the non-existent data only—thus considerably reducing their bandwidth and storage consumption.

In [7], Harnik *et al.* describe a number of threats posed by client-side data deduplication, in which an adversary can learn if a file is already stored in a particular cloud by guessing the hashes of predictable messages. This leakage can be countered using Proofs of Ownership schemes (PoW), which enable a client to prove it possesses the file in its entirety. PoW are inspired by Proofs of Retrievability and Data Possession (POR/PDP) schemes [8], with the difference that PoW do not have a pre-processing step at setup time. The literature features a number of PoW-based solutions leveraging Merkle trees, Bloom filters, etc.

Even if the privacy leakage associated with data deduplication can be addressed by relying on PoW schemes, data deduplication nevertheless conflicts with one of the most important cloud security requirement: *data confidentiality.* Recall that data confidentiality is generally satisfied through encryption. However, existing encryption solutions are incompatible with data deduplication since two identical data segments belonging to two different users become indistinguishable once they are encrypted—and hence cannot be deduplicated anymore.

In this section, we analyze this tension and discuss a number of possible avenues to reconcile confidentiality with efficiency for multi-tenant cloud storage services.

2.1 Message-Locked Encryption: The Good and The Bad

With the rise of privacy concerns, data encryption becomes compulsory for cloud storage solutions. As mentioned earlier, current encryption schemes however come at odds with the multi-tenant cloud storage technology which relies on the use of data deduplication techniques to maximise space savings. For instance, two tenants T_1 and T_2 wishing to upload the same file F to an untrusted cloud server, initially encrypt the data with their own keying material K_1 and K_2, respectively. When using randomized encryption, such an operation results in two different ciphertexts C_1 and C_2, and therefore renders deduplication ineffective.

The most prominent proposal to address confidentiality and storage efficiency is to rely on deterministic Message-Locked Encryption (MLE). Message-locked encryption was first formalized in [4]; an MLE scheme is a tuple $(\mathcal{P}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ of algorithms, where $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ are deterministic algorithms. Parameter generation algorithm $\mathcal{P}$ outputs a public parameter P—common to all users of the system. To encrypt a message M, the user first needs to generate a message derived key $K \leftarrow \mathcal{K}(P, M)$, and further computes the corresponding ciphertext $C \leftarrow \mathcal{E}(P, K, M)$. C can be decrypted into $M \leftarrow \mathcal{D}(P, C, K)$. In MLE, security requires that no efficient attacker can distinguish ciphertexts of unpredictable messages from random strings except with negligible probability.

The most prominent instantiation of MLE is *convergent*

	Storage efficiency through data reduction	Multi-tenancy and Resource sharing
Data encryption	Deduplication over encrypted data (Section 2)	No conflicts
Access control	Proofs of ownership (Section 2)	Shared ownership (Section 4)
Resource isolation	Secure reference monitor implementation (Section 3)	

Table 1: Requirements for efficient and secure cloud storage.

encryption [6] which encrypts data with a unique key that is derived from the data itself. Here, tenants first compute the encryption key by hashing the data segment and further encrypt this data with the computed key using a deterministic encryption algorithm. Similar to other MLE instantiations, convergent encryption suffers from a number of attacks such as dictionary attacks [7]; here, if the cloud server can predict the content of a message/file, then it can derive the corresponding encryption key and decrypt the message.

Therefore, the challenge lies in encrypting data segments deterministically with the same encryption key, while, on the other hand, preventing the cloud server from discovering this key—even when the message content are predictable.

2.2 Towards Secure Data Deduplication

A number of solutions for secure data deduplication in the cloud [2,3,13,18] recently emerged with the goal of enhancing the provisions of message-locked encryption schemes. Most of these solutions rely on the existence of an additional party that assists in the key generation phase, or even performs data encryption. In what follows, we discuss these solutions in greater details.

Popularity-based Solutions

Recall that the use of semantically secure encryption effectively prevents the detection of duplicate copies and hence restrains the deduplication ratio. To reconcile data deduplication with data encryption, recent contributions [13, 18] propose the reliance on different encryption techniques based on the *popularity* of the data. Namely, they assume that whenever a data segment is shared among more than t users (t being large), such "popular" data may not be considered as being sensitive, and as such can be encrypted with convergent encryption. On the other hand, less popular data, such as users' personal data, is likely to remain unique; such data can be encrypted using semantically secure encryption with a user-generated pseudo-random key. With time, whenever *unpopular* data becomes *popular*, the encryption could be converted to its convergent encrypted variant.

For example, Stanek *et al.* initially propose [18] to upload users' data encrypted with two layers of encryption: convergent encryption and threshold encryption. Whenever data becomes *popular* the outer layer (i.e., threshold encryption) is peeled off with the aid of a trusted third party—which allows it to be effectively deduplicated owing to the reliance on convergent encryption in the inner layer. Notice that this solution requires the client to upload another version of the same data—encrypted with a symmetric encryption solution—to cater for the case where the data remains *unpopular*. These semantically secure copies are removed from the server whenever the popularity threshold is reached.

In order to avoid the need to store two encrypted copies of *unpopular* data for the same user, *PerfectDedup* [13] further assists the user in discovering the popularity of the data prior to choosing the appropriate encryption mechanism. This process unfolds as follows: the user first sends a lookup request for the convergent encrypted version of the data. If the lookup succeeds, then the user learns that this data segment is popular and is subject to deduplication; on the other hand, if the convergent encrypted version does not exist, the user will encrypt it with a symmetric encryption scheme and upload it to the server. To counter dictionary attacks on the lookup process, PerfectDedup leverages a privacy-preserving popularity detection mechanism that relies on perfect hashing which yields well-distributed collisions for *unpopular* data. Thanks to this primitive, the user can decide which encryption solution to use to protect her data prior to its upload, without leaking any meaningful information to the untrusted cloud server. Compared to [18], PerfectDedup significantly reduces the storage and communication overhead by storing a single copy of each data segment being it *popular* or *unpopular*; the computational overhead is also optimized due to the use of symmetric encryption instead of a threshold encryption.

Notice that both solutions however rely on a third party, dubbed *index server*, in order to follow the popularity of each data segment. Indeed, since the cloud server cannot distinguish similar *unpopular* data, the *index server* keeps track of data uploads and counts duplicate data. For example, in [13], whenever a user discovers that her data is not *popular* yet, she contacts the index server who increases the popularity count for this particular data.

Solutions based on Server-aided Key Generation

Another approach of protecting deduplicated data is to protect the secrecy of the message-locked encryption key itself. A number of solutions, such as [2,3] generate this encryption key based on the data *and* a common secret key generated by an assisting trusted key server using a verifiable oblivious key generation protocol. For instance, in [3], a user executes an oblivious pseudo-random function (OPRF) protocol based on blind RSA signatures with the key server in order to generate the encryption key. Although the user's input to this protocol is a message-locked key (the hash of the data to be uploaded), the key server will contribute with its own secret key. As a result, the proposed scheme obstructs offline dictionary attacks by forcing the cloud server to contact the key server whenever it makes a guess for an uploaded file.

The benefits of the key generation module of [2, 3] are twofold:

- Since the protocol is oblivious, it ensures that the cloud provider does not learn any information about the files (e.g., about the file hash) during the process. On the other hand, this protocol enables the client to check the correctness of the computation performed by the cloud provider (i.e., verify the cloud provider's signature).
- By involving the key server in the key generation module, brute-force attacks on predictable messages can be slowed down by rate-limiting key-generation requests.

Building on the same idea, Armknecht *et al.* [2] suggest

ClearBox, which unlike [3], implements an OPRF based on blind BLS signatures. The rationale here is that although the verification of BLS signatures is more expensive than its RSA counterpart, BLS signatures are considerably shorter than RSA signatures, and are faster to compute by the key server. This, in turn, improves the scalability of the key server (w.r.t. the number of keys generated per second). In addition, ClearBox leverages novel cryptographic accumulators in order to allow a storage service provider to transparently attest to its customers the deduplication patterns of the (encrypted) data that it is storing. By doing so, ClearBox enables cloud users to verify the effective storage space that their data is occupying in the cloud, and consequently to check whether they qualify for benefits such as price reductions, etc.

ClearBox can be integrated with existing cloud storage providers such as Amazon S3 and Dropbox without any modifications, and motivates a new cloud pricing model which takes into account the level of deduplication undergone by data. Such a model does not threaten the profitability of the cloud business and—on the contrary—gives considerable incentives for users to store large and popular data such as music and video files in the cloud (since the storage costs of popular data might be cheaper).

Solutions based on User Collaboration

Notice that solutions based on server-aided key generation do not prevent a curious key server from performing brute-force searches on predictable messages, acquiring the hash, and the corresponding key. In this sense, the security offered by such schemes in the presence of a curious key server reduces to that of existing MLE schemes. To remedy this issue, Liu *et al.* [10] suggest a solution that does not rely on any independent server but on the collaboration of the users uploading the same file. Indeed, a cloud user encrypts a file with the same encryption key that was used by previous uploaders of the same file. Owing to the use of an additively homomorphic encryption, password-authenticated key exchange (PAKE), and a short hash function, the solution achieves deduplication with better security guarantees compared to previous solutions. More specifically, whenever a file is uploaded for the first time, the data owner also uploads a short hash of its cleartext version (10-20 bits long). The cloud server pairs this hash value with a list of all users whose files map to it. Given this, a data uploader can detect the potential data owners of its to-be-deduplicated data without revealing the cleartext version to the cloud. The uploader further runs a single round PAKE protocol together with each of these cloud users whereby the password is set as the (low-entropy) hash of their files. The protocol delivers to the new uploader the actual file encryption key whenever both files are identical, and a random key otherwise. By leveraging the PAKE-based protocol, the proposed scheme prevents dictionary attacks without the aid of any additional server.

Summary

Table 2 summarizes our findings. Namely, existing solutions either require third parties (e.g., assisting service or index server) to ensure resilience to dictionary attacks (only against the cloud provider), or require online user presence to prevent dictionary attacks. Devising secure deduplication technologies that achieve full resistance against dictionary attacks without requiring online user presence thus emerges as an interesting research direction.

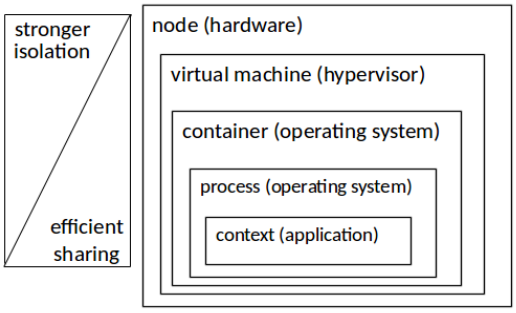

Figure 2: Possibilities for sharing and isolating computational resources. Isolation is strongest on dedicated hardware, while sharing is most efficient within an application.

3. MULTI-TENANCY VS. RESOURCE ISOLATION

In addition to the existing conflict between data confidentiality and storage efficiency (cf. Section 2), cloud providers that offer multi-tenancy solutions face another inevitable dilemma. On the one hand, they have to ensure that tenants' resources are well isolated from one another, because a breach in resource isolation can result in information leakage or even compromise. On the other hand, this contradicts a cloud provider's incentive of making efficient use of their resources—resources that are not shared might lie idle, such as a CPU core that is not in use or a disk that holds only a small amount of data.

Sharing can be implemented for different resources, at different granularities and in different layers of the software stack. In a multi-tenant cloud scenario, we are primarily interested in computing, network, and storage resources. From bottom to top, those resources can be shared at the hardware, the hypervisor, the operating system, or the application layer. In the context of computational resources, the sharing granularity ranges from a tenant being assigned a dedicated hardware node, a dedicated virtual machine within a node, a dedicated container within an OS, a dedicated process within an OS, or just a dedicated context within an application (Figure 2).

Clearly, the sharing of resources needs to be mediated by a privileged entity. This entity, which is traditionally referred to as the *Reference Monitor*, has the necessary privileges to access all tenant resources, and is in charge of performing authentication and authorization of tenants. On top of access control, the reference monitor needs to isolate the resources sufficiently such that only entitled tenants can access a requested resource. For instance, when sharing computational resources at the application layer, the application logic needs to ensure that tenants are not able to access another tenant's memory region. Often the reference monitor will work together with features of the underlying hardware to implement resource isolation. For example, CPUs offer the possibility to run code in different privilege levels (rings) and virtual memory is enabled by the memory management unit.

From a security point of view, the degree to which re-

	Resilience to Dictionary Attacks	Requires online users	Reliance on third parties
Convergent Encryption	No	No	No
Popularity-based Solutions	Only for unpopular data	No	Requires an index server
Server-assisted Solutions	Only against the cloud storage	No	Requires an assisting server
Solutions based on user collaboration	Yes	Yes	No

Table 2: Comparison of existing secure deduplication techniques.

sources are shared becomes a pivotal aspect. The lower the level of sharing, the smaller is the privileged code base, yet at the same time the more severe are the consequences of a bypass. For example, a bypass at the process level, will just be able to go as far as the privileges of the process reach, e.g., on Unix systems it will only be able to access memory of processes within the same group – but the attack surface is potentially relatively large, including the whole operating system. On the other hand, hypervisors with their typically small code base expose little attack surface, yet a bypass allows access to the actual hardware.

3.1 Vulnerabilities, Misconfigurations and Side Channels

Building a secure multi-tenant solution entails multiple challenges. Foremost, the solution should provide each tenant with a software and hardware stack that appears to be exclusive per tenant. This goal needs to be well-balanced with resource isolation at a level that maximizes resource sharing for the cloud provider. Finally, resource isolation needs to be implemented in a manner such that it cannot be bypassed.

A fundamental problem is that the reference monitor's own code is often vulnerable, e.g., to memory corruption attacks. Privileged components tend to grow in size as they become widely used—which increases the number of vulnerabilities that could be exploited by attackers to escalate privileges. For instance, OS kernels are reference monitors for enforcing discretionary-access-control-based process separation. However, commodity operating systems such as Linux have grown to sizes in the order of tens of millions of lines of code over the year, and have dozens of vulnerabilities found each year. Out of these vulnerabilities, a few can be exploited to elevate privileges and completely bypass the isolation enforced by the OS. Another example are runtime environments such as Java virtual machine implementations. The security properties of managed code, which can be helpful in writing secure multi-tenant applications, are enforced by the JVM—vulnerabilities in the execution environment allow for bypasses.

Notice that software vulnerabilities are not the sole reason for weak resource isolation. Misconfigurations and lax permissions are serious threats to security, and cannot be alleviated by implementing completely bug-free resource isolation. In addition, side-channels established through co-location of two tenant's resources can leak sensitive information [14]. Side-channels can be both software- and hardware based. Memory deduplication at the hypervisor level can be leveraged to leak data across VM boundaries; at the operating system level the same issue applies to containers and processes [5]. On the hardware side, cache side channels can be used to attack cryptographic software.

3.2 Secure Resource Isolation in Multi-tenant Settings

One way is to ensure that the code that performs access control is secure, as vulnerabilities in this code would allow for access control checks to be bypassed and resource isolation to be breached. There are three steps to securing code: reducing the amount of bugs, reducing the attack surface and hardening. While testing is typically associated with the functional aspects of a program, vulnerability-targeted testing aims at revealing security-critical bugs. A popular approach is fuzz testing, feeding random input to the program to trigger memory corruption bugs. At the moment, Driller [15] is one of the state-of-the-art approaches in this area, combining fuzz testing with symbolic execution to explore and test critical parts of application code. However, cloud software stacks might be well beyond what application-level fuzz testing tools are capable of. For instance, native distributed systems code that runs in a privileged mode is less than straightforward to instrument and binary non-standard communication protocols often do not come with extensive test suites that would be required for efficient testing.

In addition to testing, the attack surface of the code, i.e., the code paths traversable by an attacker, should be reduced as much as possible. Applications running on top of a cloud software stack might only use a subset of the software stack's capabilities. There are two important aspects to why reducing attack surface is desirable: On one hand, the unused code might provide functionality that is useful to an attacker, making exploitation easier. On the other hand, unused code might contain vulnerabilities that allow for a successful exploit in the first place. Serving as the reference monitor for both process isolation as well as container isolation, the Linux kernel is an important piece of software. Ktrim [9] performs attack surface reduction on the Linux kernel for a given application.

Since neither testing nor attack surface reduction are truly effective in deterring attacks, the exploitation of the remaining vulnerabilities needs to be mitigated through code hardening. In addition to well-established techniques such as ASLR, which has been improved over the years to the point where research prototypes apply it at runtime, software fault isolation instruments code to contain the impact of exploits and has been employed to sandbox browser extensions [20]. Further, recent advances in control flow integrity protection are now available in compilers [19]. Unfortunately the more elaborate hardening techniques such as CFI tend to cause a non-negligible performance overhead. Approaches to cut these overheads down by lowering the precision of those methods have been shown to be misguided, as the resulting implementations were too ineffective. Instead, we believe that knowing the attack surface is key to efficient hardening, because precise and effective approaches can be selectively applied to those parts of the code that actually process attacker-controlled data.

An orthogonal building block to achieve secure resource isolation is encryption. While not being able to prevent an

	Low Sharing Degree	Frequent Migration	Encryption	Vulnerability-targeted Testing	Attack-Surface Hardening
Side Channels	Yes	Yes	Yes	No	No
Privilege Escalation	Yes	No	No	Yes	Yes

Table 3: Threats to resource isolation and the effectiveness of possible countermeasures.

attacker from corrupting or deleting data on its own, it can be used to keep data confidential. This does not only apply to data at rest: In combination with computation, PrivExec has shown how encryption can be used to prevent leakage of process runtime data by encrypting it whenever memory is paged out to disk [12]. The most thorough solution that combines encryption and access control for private execution is the recently released Intel SGX. With SGX, so-called enclaves are isolated environments in which code can be executed in a completely private manner. Through CPU and MMU hardware support, the memory used by enclaves is encrypted and not directly accessible from outside an enclave, including the operating system. Hardware support for memory isolation has also been implemented by ARM's memory domains and been subject to recent academic research [16].

Aside from specific solutions, side channel mitigation largely remains an open problem. For example, hardware that combines isolation with encryption can mitigate side channels, yet software solutions that build upon these features are limited to this specific hardware. One generic mitigation of side channels in the cloud is frequent migration, i.e. switching to a separate set of the underlying resources. An example is moving VMs from one physical host to another in short intervals.

Finally, choosing the right methods to achieve efficient resource isolation depends on the task at hand. For example containers might be the right pick for scenarios that require multiple process groups to have separate namespaces, but potentially add unnecessary complexity where resource isolation can be better performed by an application's runtime.

Summary

Table 3 summarizes our findings. The biggest threats to secure resource isolation are vulnerabilities in the reference monitor and side channels. Clearly, the best countermeasure—isolating resources at a very low level using dedicated hardware—is not desirable. Side channels can be mitigated by frequent migration, but in practice this approach requires multiple physical hosts and sufficiently small transfer intervals. This makes encryption—potentially enabled by hardware such as SGX—a suitable countermeasure against side channels. To build a secure reference monitor, privilege escalation vulnerabilities need to be avoided. Vulnerability-targeted testing is essential, yet current solutions do not scale to distributed software stacks. Combining the use of hardening measures with attack-surface reduction ensures that the performance overhead of powerful hardening mechanisms remains low—even for reference monitors with a large code base such as OS kernels.

4. RESOURCE ISOLATION VS. DATA OWNERSHIP

Tenant isolation, due to its nature, hinders collaboration between the tenants. Cloud services, however, are often utilized to bootstrap collaboration platforms. Clearly, in order to collaborate, tenants should be able to interact and exchange data while remaining isolated from the other tenants. This creates a tension between collaboration and isolation—which might lead to shortcomings with respect to functionality or security.

One example for the need of collaboration is the notion of shared file ownership [17]. As cloud platforms allow collaboration on files, they should also allow collaborative access control decisions on these files. In contrast, if just one tenant hosts the collaboration account, this tenant is the sole data owner who unilaterally makes access control decisions.

There are two main arguments why shared ownership may be preferred to individual ownership in existing clouds. First, a sole owner can abuse his rights by unilaterally making access control decisions. The community features a number of anecdotes where malicious users revoke access to shared files from other collaborators. Second, even if tenants are willing to elect and trust one of them to make access control decisions, the elected owner may not want to be held accountable for collecting and correctly evaluating other tenants' policies. For example, incorrect evaluations may incur a negative reputation or financial penalties.

4.1 The Notion of Shared Ownership

To define the notion of shared ownership, Soriente *et al.* introduce the shared ownership access control model, called SOM [17]. In SOM, certain tenants become *owners*. These owners have to approve every file operation. An operation is approved if at least t owners support it. An overview of SOM is shown in Figure 3.

Since cloud platforms currently do not support shared ownership, the tenants have to achieve the notion of shared ownership using existing access control mechanisms. The supported access control policies focus on providing isolation; isolation is of paramount importance since secret/sensitive material should not leak to unauthorized tenants.

SOM requires different kinds of isolation to protect against different adversaries. For instance, performing access control using data encryption alone is not sufficient, since shared ownership demonstrates the need for higher-level isolation mechanisms. That is, shared ownership should be resistant to key leakage and collusion attempts. If the file key is (inadvertently) leaked by one file owner, a user without read permission should not be able to read the corresponding file.

4.2 Enforcing Shared Ownership in the Cloud

In order to instantiate SOM and achieve shared ownership, centralized enforcement is not well suited, as it would depend on a central Policy Decision Point (PDP). The tenant controlling this PDP could make unilateral access control decisions, which conflicts our previously outlined isolation requirements. In practice, the shared ownership policy agreed upon by the owners could always be bypassed, and thus the notion of shared ownership nullified.

Any effective enforcement solution should be distributed, since it must grant an operation if and only if t owners separately support the grant decision. To effectively realize distributed enforcement with isolation guarantees, owners have

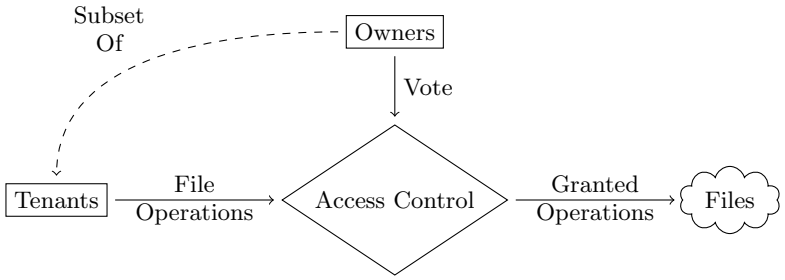

Figure 3: Example of shared ownership in the cloud. Here, a subset of the tenants votes to jointly make access control decisions on shared files.

separate accounts on potentially different cloud platforms. These cloud platforms do not allow the deployment of custom enforcement components, which makes the enforcement even more difficult. Instead, cloud platforms only support basic access control policies via Access Control Lists (ACLs).

To this end, Soriente *et al.* [17] propose, Commune, the first concrete proposal to instantiate shared ownership atop agnostic cloud platforms. Commune supports read and write operations on files. To write a file to the shared repository, the writer encodes the file into tokens and distributes the tokens to the owners' accounts. A file is written to the shared repository if and only if the writer successfully distributes the file's tokens onto at least t owners' accounts. That is, a write access granted to the shared repository is equivalent to write access to at least t of the owners' accounts. Read access works analogously. Such basic access control policies can be implemented using ACLs.

To protect tokens' confidentiality from curious cloud providers, Commune proposes to encrypt the tokens using a collusion-resistant secret sharing scheme. Namely, to isolate tenants from each other and thereby protect the system from collusion, the key shares are cryptographically bound to the target tenant and cannot be combined with shares of another tenant. This is due to the fact that each key share is blinded by the user ID and the blinding can only be removed by combining the correct shares.

Commune additionally enforces shared ownership as follows: (any parts of the) file contents can be recovered if and only if t file tokens and t key shares are available. In particular, given $t-1$ file tokens and t key shares or t file tokens and $t-1$ key shares, no file contents can be recovered. To achieve this property, the file tokens are encoded using a special all-or-nothing encoding, which cryptographically links the different tokens with each other through an encryption scheme.

Summary

Table 4 summarizes our findings. Namely, previous solutions do not provide all the required isolation guarantees. We require isolation against a curious server (confidentiality), against a misbehaving member (revocation), against partially authorized, colluding members (collusion resistance) and against an unfair administration (fairness). The work in [17] is able to satisfy all the isolation requirements, while providing the functional requirements based on agnostic cloud platforms that only offer *basic* access control functionalities. Extending Commune [17] to support a richer set of access control operations and improving the overall system performance emerges as an interesting research problem.

5. OUTLOOK

In this paper, we explored the solution space to reconcile security and functional requirements in the cloud. In this respect, we described a number of solutions that achieve data confidentiality in the cloud—while supporting deduplication. We also reviewed and analyzed existing techniques for isolating resources in multi-tenant cloud environments. Based on this analysis, we postulate that existing cloud storage platforms are still too weak when it comes to isolating tenants and containing attacks, and argue that the threat of unknown vulnerabilities and the subsequent loss of data governance is still one of the main reasons why businesses still shy away from using the cloud. Yet, without resource sharing, the cloud model cannot be successfully implemented. Indeed, we motivate this tension between resource sharing and isolation by introducing the notion of shared ownership, and we discuss a number of practical solutions to bootstrap shared ownership within existing cloud storage platforms.

We still expect other hazards to arise with respect to designing storage-efficient cloud security primitives for multi-tenant settings. For instance, cloud customers typically require some guarantees on the integrity and retrievability of their data—which can be ensured through recently designed solutions called proofs of retrievability [8]. Existing PoR solutions assume a single-tenant setting, and as such cannot be directly integrated in a multi-tenant cloud environment that enforces data deduplication. This clearly hinders the large-scale adoption of PoR solutions by cloud providers.

Given the current trends in designing secure cloud solutions, our findings motivate the need to consider functional requirements, such as multi-tenancy, data deduplication, resource sharing, in all aspects of the design of security primitives for the cloud. We can identify a number of reasons why it might be beneficial to embed such functional requirements in the design of future security primitives: *(i)* supporting functional requirements would only increase the adoption and integration of cloud security primitives within existing cloud platforms at marginal costs, *(ii)* this, in turn, would lay down the basic foundations to secure cloud-based businesses, and protect users from surveillance, loss or exposure of sensitive data. We therefore hope that our findings motivate further research in this area.

	Confidentiality	Revocation	Collusion Resistance	Fairness	Efficiency
Centrally-managed Repository	Yes	Yes	Yes	No	Yes
Cloud Storage with ACLs	Yes	Yes	No	No	Yes
Commune [17]	Yes	Yes	Yes	Yes	No

Table 4: Comparison of existing collaborative storage techniques.

Acknowledgements

The authors would like to thank the anonymous reviewers for their valuable feedback and comments. This work was partly supported by the TREDISEC project (G.A. no 644412), funded by the European Union (EU) under the Information and Communication Technologies (ICT) theme of the Horizon 2020 (H2020) research and innovation programme.

6. REFERENCES

[1] Armknecht, F., Bohli, J.-M., Karame, G. O., Liu, Z., and Reuter, C. A. Outsourced Proofs of Retrievability. In *Proceedings of ACM SIGSAC Conference on Computer and Communications Security (CCS)* (2014).

[2] Armknecht, F., Bohli, J.-M., Karame, G. O., and Youssef, F. Transparent data deduplication in the cloud. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2015), CCS '15, ACM, pp. 886–900.

[3] Bellare, M., Keelveedhi, S., and Ristenpart, T. DupLESS: Server-aided Encryption for Deduplicated Storage. In *Proceedings of the 22Nd USENIX Conference on Security (USENIX SEC)* (2013), pp. 179–194.

[4] Bellare, M., Keelveedhi, S., and Ristenpart, T. *Message-Locked Encryption and Secure Deduplication.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 296–312.

[5] Bosman, E., Razavi, K., Bos, H., and Giuffrida, C. Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector. In *Proceedings of the IEEE Symposium on Security and Privacy* (2016).

[6] Douceur, J. R., Adya, A., Bolosky, W. J., Simon, D., and Theimer, M. Reclaiming Space from Duplicate Files in a Serverless Distributed File System. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS)* (2002).

[7] Harnik, D., Pinkas, B., and Shulman-Peleg, A. Side channels in cloud services: Deduplication in cloud storage. In *Proceedings of the 4th ACM International Wokshop on Storage Security and Survivability (StorageSS)* (2008).

[8] Juels, A., and Jr., B. S. K. PoRs: Proofs of Retrievability for large files. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)* (2007), pp. 584–597.

[9] Kurmus, A., Sorniotti, A., and Kapitza, R. Attack Surface Reduction for Commodity OS Kernels: Trimmed Garden Plants May Attract Less Bugs. In *Proceedings of the European Workshop on System Security* (2011), ACM.

[10] Liu, J., Asokan, N., and Pinkas, B. Secure Deduplication of Encrypted Data Without Additional Independent Servers. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security (CCS)* (2015), pp. 874–885.

[11] Meyer, D. T., and Bolosky, W. J. A Study of Practical Deduplication. In *Proceedings of the 9th USENIX Conference on File and Stroage Technologies* (Berkeley, CA, USA, 2011), FAST'11, USENIX Association, pp. 1–1.

[12] Onarlioglu, K., Mulliner, C., Robertson, W., and Kirda, E. PrivExec: Private Execution As an Operating System Service. In *Proceedings of the IEEE Symposium on Security and Privacy* (2013).

[13] Puzio, P., Molva, R., Önen, M., and Loureiro, S. PerfectDedup: Secure data deduplication. In *10th International Workshop on Data Privacy Management (DPM)* (2015).

[14] Ristenpart, T., Tromer, E., Shacham, H., and Savage, S. Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds. In *Proceedings of the ACM Conference on Computer and Communications Security* (2009), ACM.

[15] Shoshitaishvili, Y., Wang, R., Salls, C., Stephens, N., Polino, M., Dutcher, A., Grosen, J., Feng, S., Hauser, C., Kruegel, C., and Vigna, G. Driller: Augmenting Fuzzing Through Selective Symbolic Execution. In *Proceedings of the Network and Distributed System Security Symposium* (2016).

[16] Song, C., Moon, H., Alam, M., Yun, I., Lee, B., Kim, T., Lee, W., and Paek, Y. HDFI: Hardware-Assisted Data-flow Isolation. In *Proceedings of the IEEE Symposium on Security and Privacy* (2016).

[17] Soriente, C., Karame, G. O., Ritzdorf, H., Marinovic, S., and Capkun, S. Commune: Shared ownership in an agnostic cloud. In *Proceedings of the 20th ACM Symposium on Access Control Models and Technologies* (New York, NY, USA, 2015), SACMAT '15, ACM, pp. 39–50.

[18] Stanek, J., Sorniotti, A., Androulaki, E., and Kencl, L. A Secure Data Deduplication Scheme for Cloud Storage. In *18th International Conference on Financial Cryptography and Data Security (FC)* (2014), pp. 99–118.

[19] Tice, C., Roeder, T., Collingbourne, P., Checkoway, S., Erlingsson, Ú., Lozano, L., and Pike, G. Enforcing Forward-Edge Control-Flow Integrity in GCC & LLVM. In *USENIX Security Symposium* (2014), USENIX Association.

[20] Yee, B., Sehr, D., Dardyk, G., Chen, J. B., Muth, R., Ormandy, T., Okasaka, S., Narula, N., and Fullagar, N. Native Client: A Sandbox for

Portable, Untrusted x86 Native Code. In *Proceedings of the IEEE Symposium on Security and Privacy* (2009).