An Overview of Issues and Recent Developments in Cloud Computing and Storage Security

Everaldo Aguiar, Yihua Zhang, and Marina Blanton

Abstract The recent rapid growth in the availability and popularity of cloud services allows for convenient on demand remote storage and computation. Security and privacy concerns, however, are among the top impediments standing in the way of wider adoption of cloud technologies. That is, in addition to the new security threats that emerge with the adoption of new cloud technology, a lack of direct control over one's data or computation demands new techniques for service provider's transparency and accountability. The goal of this chapter is to provide a broad overview of recent literature covering various aspects of cloud security. We describe recently discovered attacks on cloud providers and their countermeasures, as well as protection mechanisms that aim at improving privacy and integrity of client's data and computations. The topics covered in this survey include authentication, virtualization, availability, accountability, and privacy and integrity of remote storage and computation.

1 Introduction

Cloud computing enables on-demand access to computing and data storage resources that can be configured to meet unique constraints of the clients with minimal management overhead. The recent rise in the availability of cloud services makes them attractive and economically sensible for clients with limited computing or storage resources who are unwilling or unable to procure and maintain their own computing infrastructure. The ever increasing need for computing power and storage

Everaldo Aguiar
University of Notre Dame, Notre Dame, IN, e-mail: eaguiar@nd.edu

Yihua Zhang
University of Notre Dame, Notre Dame, IN, e-mail: yzhang16@nd.edu

Marina Blanton
University of Notre Dame, Notre Dame, IN, e-mail: mblanton@nd.edu

accounts for the steady growth in popularity of companies offering cloud services. Clients can easily outsource large amounts of data and computation to remote locations, as well as run applications directly from the cloud.

As an example, Chow et al. [21] point out that some of the world's largest software companies today operate entirely from the cloud or at least have a major portion of their services outsourced to a cloud environment. With all the convenience being offered for a relatively low price, one would expect that most, if not all, companies would migrate to the cloud. There are, however, impediments to this transition, with security commonly cited as the number one concern by both the private sector and the government [33, 72]. Public clouds have not yet reached a level of security that enables private users and corporations to entrust them with sensitive data. As a concept that was created from a combination of several building blocks, cloud computing naturally inherited the security flaws present in each component. In addition, new issues that did not previously exist arose with this new technology trend. The evidence for this problematic side of cloud computing can be seen in the considerable literature [1, 20, 21, 42, 49, 58, 60, 62, 75] that surveys several of the general and specific security issues that affect clouds. Cloud security is also the focus of this work.

Unlike prior surveys of cloud security issues, our ultimate goal is to provide a much more complete and thorough coverage of the research literature related to this topic. We give a broad overview of publications in the fields of cloud computing security and security of remote storage and computation. In particular, the topics covered in this work include:

- *Client authentication and authorization:* We cover the current body of work on methods for disrupting and exploiting the interface between a cloud provider and its clients, usually carried out via a web browser.
- *Security shortcomings of hardware virtualization:* We describe the problems that have surfaced along with the massive use of hardware virtualization by cloud providers. We indicate how virtualization can be exploited to obtain unauthorized information from vulnerable users, and also indicate mitigation techniques that can be employed. In addition, we also address vulnerabilities related to the usage and sharing of virtual machine (VM) images.
- *Flooding attacks and denial of service (DoS):* Because cloud computing systems are designed to scale according to the demand for resources, an attacker may use that characteristic to maliciously centralize large portions of the cloud's computing power, lowering the quality of service that the cloud provides to other concurrent users. We discuss different types of attacks on cloud availability and their potential consequences.
- *Cloud accountability, or its ability to capture and expose wrongful activity:* We discuss capabilities that an accountable system should have and solutions for achieving these capabilities.
- *Challenges and solutions for remote storage protection:* We describe several techniques that can be employed by cloud clients to verify integrity of their outsourced data.

- *Protection of outsourced computation:* Finally, we give an overview of current approaches for assuring privacy and integrity of outsourced computations.

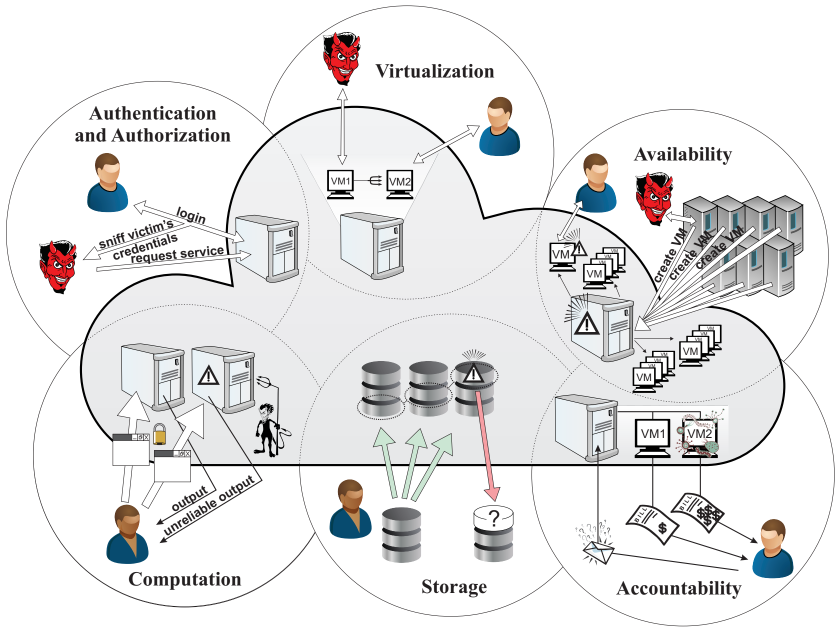

Fig. 1 Overview of the topics covered in this article

Fig. 1 provides a summary of the topics covered in this work.

Before proceeding with the coverage of the above mentioned topics, we define some of the common terminology used in the context of clouds. Although the members of the cloud community in its early days did not all agree on which features should actually be part of the broad concept of cloud computing, the definition that was later provided by the U.S. National Institute of Standards and Technology (NIST) [58] precisely subdivided cloud computing into three distinct models, which offer differing capabilities to the consumer.

- *Software as a Service (SaaS):* Following this model, the cloud service provider makes both the software and the cloud infrastructure to run it available to the client, while it retains complete control over the underlying physical settings of the cloud (i.e., the operating system, network, storage, etc.) and the individual application capabilities. Thin client interfaces such as web browsers are often used to allow access to these applications.
- *Platform as a Service (PaaS):* Similar to the previous model, PaaS aims at giving the client the ability to run software on the cloud infrastructure. However, in this case the consumer can deploy his own applications or create personalized ones by using tools supported by the cloud provider. Control over the physical settings

remains restricted to the cloud provider but the clients of this model are able to fully manage the deployed applications.
- *Infrastructure as a Service (IaaS):* The consumers of this model can provision computing resources including storage, processing, and networks, which can usually be used to host any arbitrary operating system and applications. Most of the well established cloud services (such as Amazon EC2) adopt this model and charge their users according to the amount of utilized resources.

2 Authentication and Authorization

2.1 Browser security

One of the goals of providers of Software as a Service (SaaS), as pointed out in [47], is to have a generic means of offering access to their clients, independent of their local platform. Internet browsers are the ideal choice and can be used on the client side to simply perform I/O, while the actual computation takes place in the cloud back-end. As a downside, however, known browser security issues may affect this interface for cloud authentication. We review two issues that can be exploited in this context:

- *XML-based authentication in a browser:* User authentication to the cloud via a web browser requires the issuing of cryptographically valid Extensible Markup Language (XML) tokens. As the first step of the process, the user provides his login credentials, which upon verification and validation, allow the cloud to issue the security tokens which the client will later provide to the cloud server as means of authentication. This process, however, requires the assistance of a trusted third party as browsers are not capable of issuing XML-based security tokens on their own. There are multiple services that can be used as an intermediary between users and the cloud, including Microsoft's Passport, which Jensen et al. [47] use to illustrate the security shortcomings of this interface. In that scenario, when the user wishes to authenticate to the cloud, an Hypertext Transfer Protocol (HTTP) redirect is sent to the Passport login server. After the user enters information, the server will translate those credentials into a Kerberos token which is then, via another HTTP redirect, sent to the cloud server, finally allowing the user to be authenticated. The downside of this approach lies in the mechanism via which the tokens are made secure. The Kerberos tokens sent by Microsoft Passport to the cloud server are only protected by the Same Origin Policy (SOP), which has been shown to be vulnerable [91]. Hence, if an adversary gains control of these tokens, he will be granted full access to all of his victim's services. To counteract this problem, the paper [47] discusses a few methods via which XML tokens can be made safe with the use of Transport Layer Security (TLS). This paper also suggests that a more optimal solution to these limitations would be to further enhance the capability of web browsers, adding `XML Encryption` and `XML`

`Signature` to the browser security API. In [57], a widely known attack on XML signatures was described in which the attacker intercepts a given Simple Object Access Protocol (SOAP) message being sent to the server by some client, replacing its content with some arbitrary request on behalf of the attacked client; Amazon EC2 is vulnerable to a variation of this signature wrapping attack [38].

- *Federated Identity Management based on insecure SOP:* The legacy Same Origin Policy (SOP) is mainly used for defining access rights to web scripts. Read/write operations are only permitted to requests originated from the same source, which in a Web context is usually defined jointly by the domain name, the protocol, and the port. As previously stated, the tokens used during client authentication to the cloud are often protected by this policy. Jensen et al. [47], however, point out that Domain Name Systems (DNS) can be infected in such a way that would cause domain names to become unreliable, compromising the functioning of the security policy. For instance, it has been shown that an attacker can host a malicious website, which upon being visited, will attempt to change the DNS settings of the user's broadband network router [71]. Therefore, using SOP exclusively for the purpose of protecting these authentication tokens does not strictly guarantee security. To mitigate this problem, [47] elaborates on better ways of integrating SOP and TLS to improve security of these protocols.

In order to ensure the safety of their clients' information, cloud providers have claimed to address the above forms of attack. However, a recent work by Somorovsky et al. [70] reports two successful variants of the signature wrapping attack on both the Amazon cloud services and Eucalyptus [30]. The first attack variant used a single eavesdropped SOAP request for the `MonitorInstances` operation with a duplicated body to perform arbitrary operations. The SOAP interface used by Amazon EC2 validated the XML signature for the original (unmodified) SOAP body, but performed the operation described in the maliciously injected body. To circumvent the timestamp verification performed before the execution of the SOAP messages, the authors similarly duplicated the timestamp element in the security header and were able to execute arbitrary operations even when the original timestamp had already expired. Upon being notified of the issue, the Amazon Web Services (AWS) security team quickly provisioned a patch that prevented the execution of messages containing duplicated timestamp elements. The second attack variant, however, showed that it was possible to issue SOAP messages with several `<soap:Body>` elements and technically achieve the same result. The modified message used in that case had three `<soap:Body>` bodies, with the second body containing the originally signed timestamp. That timestamp and the whole third body were verified by the signature verification component, while a different timestamp located in the security header was tested for expiration and the first body was interpreted to determine the operation to be performed.

The SOAP interface used by Eucalyptus is slightly different and validates the format of incoming SOAP messages against an XML schema. In such a scenario, messages containing duplicated body or timestamp elements were automatically rejected and the previously mentioned attacks were shown to be infeasible. Nevertheless, by modifying the classical signature wrapping attack [57] and by using a duplicated security header element that did not violate the XML schema, the authors were able to place the signed body and timestamp elements into a new location and replace the original ones with their own.

Besides describing these successful attack implementations, [70] also shows how advanced Cross Site Scripting (XSS) techniques can be used to steal data credentials from Amazon and Eucalyptus cloud users. In the Amazon case, Somorovsky et al. demonstrate that it is possible for an attacker to utilize the public Amazon shop discussion forums as a means for exposing user information needed to perform the previously mentioned signature wrapping attacks. The attacker simply needs to create a new discussion topic containing Java Script code to obtain cookie data via `document.cookie`. Despite the fact that Amazon utilizes padding techniques to prevent malicious code from being executed, a careful formatting of the code block was able to deceive that mechanism. Once in possession of this data, the attacker can immediately obtain access to that user's cloud services since the same log-in credentials are shared across the Amazon shop and Amazon cloud control interface.

2.2 Binding issues

In order to accurately maintain and instantiate VMs (in the case of IaaS) or specific modules (in the case of PaaS), the cloud provider needs to store metadata descriptions associated with each operation. A user may refer to these metadata descriptions in order to determine the detailed functionality of a certain service upon having a particular need. Jensen et al. [46] suggest a spoofing attack that aims at reengineering this metadata.

A *metadata spoofing attack* [46] allows an adversary to modify metadata descriptions, potentially causing severe damage to the user's services. An example of this attack given in [47] suggests that an attacker may attempt to modify the server's Web Service Definition Language (WSDL) descriptions. This can, for instance, cause a call to `deleteUser` to generate a SOAP message that mimics and may be interpreted as a call to `setAdminRights` on the server side. Therefore, in that scenario, a user who intended to delete a certain account may inadvertently grant extra privileges to the adversary.

Clouds are especially susceptible to these types of attacks because they use WSDL repositories that are dynamically utilized by all users, increasing the probability of an attacker successfully spreading his malicious WSDL file. As a countermeasure, cloud providers need to cautiously check all metadata documents for authenticity, which may be somewhat troublesome as there are no standard mechanisms for doing so. Another possible approach would be to ensure the establishment of trust relationships with users prior to accepting their requests, although this may not be applicable to some scenarios [46].

3 Hardware Virtualization-Related Issues

In order to satisfy the demand for computation from multiple concurrent users, cloud service providers often need to multiplex the use of their limited available hardware. Amazon EC2 and Microsoft Azure, for instance, allow users to dynamically instantiate VMs which can be tuned to their specific needs and easily managed while running on the top of a shared physical infrastructure. Although the general use of virtualization normally implies an isolation between the workloads involved, Ristenpart et al. [63] show that new vulnerabilities can arise when an adversary is capable of determining the location of a particular target VM in the cloud infrastructure and subsequently instantiating a malicious co-resident VM. Though not particularly trivial to exploit, these vulnerabilities can vary from the extraction of cryptographic keys to DoS, depending on the adversary's intent.

Another virtualization-related security issue, raised by Christodorescu et al. [22], is that vulnerabilities arise from the fact that in highly multiplexed cloud environments, providers often have no control over what types of VMs are being deployed by their clients. Conversely, users also have a very limited knowledge of the underlying physical infra-structure. We discuss how attackers can instantiate VMs with modified OS images, how that may constitute a threat to other users, and what solutions were put forth by [22]. Along similar lines, Wei et al. [83] explored the possible risks concerning how VM images are stored and managed in the cloud. Specifically, they showed that an attacker can build and share images that are contaminated with malware and other types of threats. A more recent work [16] gives practical examples of how these risks can be exploited on Amazon EC2 and how they should be mitigated.

We next describe the above three virtualization-related security issues in more detail.

3.1 Exploiting VM Co-residence

Ristenpart et al. [63] investigate the possible means by which a cross-VM attack – the one that originates from a given VM and targets another – may be performed. There are four distinct steps to be carried out by the adversary in the suggested attack: (1) determining where the target VM is physically located in the cloud, (2) launching VMs that will be co-resident to the targeted one, (3) verifying that the VMs involved are indeed co-resident, and (4) exploiting cross-VM information leakage. In the following we describe each of these steps.

- *Determining the target VM's location:* Using simple network probing tools, Ristenpart et al. showed that it is possible to infer considerably accurate information about the location of VMs based on their assigned IP addresses. To do so, they first sampled a large pool of IP addresses, isolated those that were responsive and associated with EC2, and used Domain Name Systems (DNS) lookups from within the Amazon EC2 network to obtain their internal IP. In parallel to that, the researchers also launched a large number of VM instances and observed that distinct IP address ranges appeared to be assigned to each different zone. Thus, they maintain that by simply having the internal IP address of a given VM, one can make fairly accurate deductions regarding its physical location. To prevent such a form of physical mapping, cloud providers need only to eliminate the static assignment of IP addresses based on availability zone, instance type, or any other feature of the VMs.

- *Instantiating co-resident VMs:* Two methods for instantiating co-resident malicious VMs are proposed by [63]. The first method consists of launching a large number of VMs in a brute-force manner until co-residency is achieved. The second method is more refined and takes advantage of the Amazon EC2 placing algorithm's tendency to assign machines belonging to a common small set to VMs being instantiated within a short period of time from each other. Deliberately targeting VMs that had been recently launched yields a successful co-residency rate of nearly one half for certain types of instances. One way to mitigate such issues, as suggested by Ristenpart et al., is to allow users to make the decision on the placement of their VMs.
- *Verifying co-residence:* Ristenpart et al. verify or discard co-residence with a high level of correctness. To assert that any two VMs are co-resident, Ristenpart et al. check that the VMs have matching Dom0 IP addresses by tracing messages sent to the target VM to see whether the addresses of the first and last hops are the same. Co-residency is also confirmed by an analysis of the round-trip times and internal IP ranges. Preventing these techniques requires configuring the Dom0 machine to ignore `traceroute` requests, randomly assigning internal IPs, or using virtual LANs to isolate accounts.
- *Exploiting cross-VM information leakage:* Assuming the attacker was able to place his VM on the same physical machine as the target instance, [63] shows that a malicious adversary may learn information about a co-resident VM via cache-based side channels (which could possibly be used to steal cryptographic keys), as well as other physical resources that are multiplexed among the co-resident instances. This work has been recently expanded by Xu et al. [87] who were able to create similar covert channels with noticeably higher bit rates than previously reported. Despite that, the later work shows that even at higher bit rates, the harm that can be caused by such techniques is limited to the extraction of very small blocks of data (e.g., private keys).

3.2 Exploiting the Cloud's Limited Knowledge of User VMs

The use of virtualization in a cloud computing environment enforces an isolation between different workloads, protecting them from other users who may attempt to compromise their information. Another security layer deals with protecting individual workloads within themselves, which is especially important when these are exposed to the Internet. Providing such protection requires a deeper knowledge of the workload, which consists of the guest OS in the case of hardware virtualization. Information such as what operating system is running in the VM, however, is often unavailable to the cloud provider as in many cases the users are allowed to upload and run their own VMs.

The most widely available techniques for virtualization-based security are designed to work when the operating system running in the VM of interest is known in advance, allowing the virtual machine monitor (VMM) to correctly track its activity. In addition, in order to assure the integrity of the supervising tool being used, these techniques require that the monitoring be initialized via a safe boot, while the guest OS is first being put to use, which obviously becomes impractical when users are allowed to start their VMs from a guest OS snapshot, as is the case in most cloud settings. To remedy this situation where a provider cannot offer appropriate security measures due to the unknown configuration and integrity of its clients' VMs, Christodorescu et al. [22] propose a new architecture based on VM introspection . The concept of VM introspection was first introduced by Garfinkel et al. [31] and more recently formalized in [61]. Through the use of such techniques, it is possible to inspect a VM from a safer location outside the monitored machine and evaluate what is happening to it internally. This mechanism works with the aid of the hypervisor, which is modified to collect data describing its interactions with the VMs being monitored and send it to safe (specially designed) VMs that will carry out the actual analysis and monitoring process.

The solution put forth by Christodorescu et al. does not rely on any assumptions about the state of the running VMs and works even when the cloud user does not give the provider any information about the software running in their VMs. Their approach requires that the guest VMs sit on the top of a correct, trusted and unbreachable hypervisor controlled by the cloud provider, who also maintains special VMs referred to as secure VMs, which are assumed to be unbreachable and host both a guest OS identification and a rootkit detection application whose functionality is based on VMI. The OS identification application uses a whitelist of known executable kernel code obtained from several different operating systems (running on clean VMs) to match and validate those originated from the monitored VM, and while doing so it can accurately determine which guest OS is running in that specific VM. Using the same principle, the rootkit detector checks for the presence of unauthorized code in kernel space, which upon detection is compared to a database (blacklist) of known malicious code for identification. Experimental tests [22] showed that these applications, running in a secure VM and using the proposed hardware-based VM introspection, were able to both identify the OS running in a monitored VM with high confidence, and detect unauthorized code running in the guest kernel space. Another important feature of this approach is that, because it uses secure introspection and runs outside the guest OS, it will still keep the system safe from rootkits (and other malware) even in the event of an attack that disables the traditional security software running in the guest.

Another work in the realm of virtualization security [74] proposed a system called NoHype, in which the need for hypervisors is eliminated in its entirety, consequently excluding the hypervisor attack surface. In order to allow VMs to run natively on the underlying hardware while still maintaining the support for multiple concurrent VMs, NoHype places guest VMs in more direct contact with the hardware to avoid indirection. It implements slight modifications to the guest OS, allowing it to perform all system discovery during bootup. In addition, CPU and physical memory need to be pre-allocated and only virtualized I/O devices are used.

3.3 Exploiting VM Images

Despite the numerous services that can be provided by clouds and the apparent complexity of their environments, the resources offered by cloud providers can be narrowed down to primarily three: (1) a pool of VMs from which users can choose the most suitable for their specific needs; (2) a set of servers that are configured to efficiently run these VMs; and in some cases (3) a means for non-volatile storage that can be used to share data across multiple VMs or simply for backup. This section has already covered the security threats related to VMs that are currently in execution and their mitigation. Persistent data storage in the cloud and the corresponding security issues are the topic of Section 6. An area that has been paid considerably less attention is that of threats that can result in compromising VM images in the cloud; this area is discussed next.

Wei et al. [83] were the first to expose the risks involved with the publishing and large scale usage of images. The authors emphasize that the sharing of VM images is of great importance since it largely simplifies administrative tasks and costs related to installing and configuring software (i.e., instead of doing all the work, a user can simply choose a VM that is preloaded with all the software that is needed). There are, however, risks to all parties involved in this process. For example, the image *publisher* may inadvertently release sensitive information while personalizing the images he wishes to share. *Retrievers* of these pre-configured images are highly vulnerable to infections that could have been loaded into the VMs by malicious publishers and which can be used, for instance, as vectors for propagating Trojan horses. Lastly, cloud *administrators*, who are entrusted with maintaining the integrity of these images, may be liable to legal action if it is shown that malicious or illegal content was associated with the images used in their system. In light of the aforementioned threats, Wei et al. proposed an image management system called Mirage that provides a series of security features. First, it offers image publishers a framework by which they can define a set of trusted users to whom they wish to allow access rights. To further protect publishers from accidentally leaking private information, Mirage also contains an elaborate image filter that removes from each image all private (e.g., passwords) and malicious (e.g., malware and pirate software) information. Next, it also contains a provenance tracking system that keeps a detailed history of each image, including all operations performed on it after its release. In addition to these features, the system is also capable of providing a set of maintenance services, such as anti-viruses, that can be periodically run over the entire pool of images. To mitigate some of the heavy overhead induced by these new features, the authors suggest running filters and services at the repository level, where scaling can more efficiently take place by taking advantage of the similarities between different images.

The filtering systems proposed in [83] are already embedded into some of today's cloud services, but their advantages have not yet outweighed the disadvantages of implementing them. This is especially true for providers not liable for issues that may occur to shared user data as in the case with Amazon EC2. A recent work [16] presented practical examples of how these vulnerabilities can be exploited and argued that the increasing competition between cloud providers may become the driving force that will finally push for a security enhancement with regards to cloud VM images. In this work, Bugiel et al. [16] successfully developed an automated tool that was able to search for and extract highly sensitive information from public Amazon Machine Images. The cost of running the attacks was very low, and the data the authors were able to obtain contained source code repositories, administrator passwords, and other types of credentials. [16] also highlighted the discovery of several widespread vulnerabilities caused by incorrect use of SSH. An alarming number of Amazon Machine Images were found to have SSH backdoors that allowed their publisher to remotely login to instances of VMs belonging to other users. Tools providing countermeasures to these threats are suggested in [83] and cloud providers can consider the trade-off between the benefits and costs of implementing these tools.

With regard to the security of VMs, we point out the work of Schiffman et al. [65], which provides an architecture that allowed for the performance of runtime integrity proofs in general purpose distributed systems. Their prototype, based on the Xen VM, uses the Clark-Wilson integrity model [24] and a main component was defined as *VM verifier*, whose job was to enforce upon running VMs the integrity requirements as defined in [24]. Though the topic of integrity protection is discussed in more depth in section 7, we briefly provide an intuition to Schiffman's solution. As a first step, the architecture monitors each VM through a secure boot process at the end of which the system's integrity is verified by a base system controlled by the provider. The provider has full knowledge of the system's integrity properties and can leverage that information to enforce VM compliance. In order to give end users complete assurance that their VMs' integrity is preserved under that architecture, providers simply need to prove that the secure initialization process was successfully completed and that the previously mentioned integrity property enforcement was carried out through runtime.

4 Availability

Among the most attractive features of cloud computing is the fact that computational power can be easily supplied on demand. If a certain service running from the cloud suffers a sudden increase in the workload, additional hardware support can be provisioned on the fly. In spite of its appeal, this “elastic” characteristic of clouds can be exploited by attackers.

Jensen et al. [47] describe how *flooding attacks* are a real threat to clouds. This issue rises in such environments when, for instance, all requests to a certain service need to be individually checked for validity, thereby causing service overloading. Consequences of service overloading pointed out by Jensen et al. are denial of service (DoS) (which may be direct or indirect) and discrepancies related to accounting and accountability; these consequences are discussed below.

- *Direct DoS:* When a service is being flooded, the cloud operating system will likely start to supply extra computational power to it. By attacking a single cloud-based address, an adversary may eventually cause a full loss of availability to the entire service if, for instance, a large enough portion of the globally available hardware becomes dedicated to the flooded application.
- *Indirect DoS:* A possible side effect of the above attack, caused by the fact that hardware is multiplexed in the cloud, is that services that are co-resident to the one being flooded may also experience similar overloading problems. Hence, although a service may not be the direct target of a flooding attack, when its available hardware is exhausted by continuous requests sent to the attacked service, it will no longer be able to perform its tasks.
- *Accounting and accountability:* Arguably the most severe consequence to the cloud user who owns the flooded service can be caused by what is thought to be another major advantage of cloud computing. Most cloud providers charge their users according to the actual usage of their infrastructure during a pre-determined time slice. In the case of a service that is being flooded, this usage will be obviously high, which, in its turn, will most likely translate to bills that are much higher than expected.

Though it may not be an easy task to remedy flooding attacks without compromising the overall performance of the cloud to some extent, Zunnurhain et al. [95] provide a feasible alternative. The authors suggest an organization that groups different servers into fleets, each of which is designated for a distinct type of job. By structuring the available hardware in this way, the provider imposes an isolation that would prevent an overload of bogus requests from affecting the performance of the computational fleet. However, a downside to this alternative is that while a certain fleet of servers may become overloaded with a large number of valid jobs, it would not be able to offload some of its tasks to other fleets that may have several idle nodes available.

A different form of a DoS attack in cloud systems is proposed and discussed by Liu [56]. The author points out that it is possible to take advantage of the fact that cloud data centers are typically under-provisioned (i.e., the hosts only use a fraction of their resources or interface speed). In the attack proposed in [56], the adversary gains control of a few hosts in a certain subnet (i.e., set of nodes connected via a common router) and then simply transmits enough traffic to hosts located elsewhere. Despite its simplicity, this attack is likely to quickly saturate the targeted network since the uplink capacity of any given router is much smaller than the aggregate uplink of the hosts in its subnet. Since uplinks are often under-provisioned in both directions, this attack will, by symmetry, also saturate the network of the subnet that is receiving the bogus traffic. Furthermore, there are usually a sizeable number of traffic bottlenecks on cloud networks, such as routers, that are connected to a large number of hosts or other routers. In addition to this, this attack can potentially be refined to aim at specific targets. In the case of a targeted attack, the adversary needs to gain control of hosts located within the same subnet of the service that he wishes to bring down.

Due to the large size of cloud data centers, which can cause them to be significantly under-provisioned, Liu [56] asserts that this attack can be particularly harmful when targeted at this type of environment. Other components that allow for the effectiveness of the proposed attack are the fact that cloud data centers are concurrently used by many different organizations and that these users do not have control over the underlying network, making them incapable of providing countermeasures when needed. Liu also proposes a couple of prevention strategies to an attack that targets specific hosts. One involves a fast dynamic migration architecture that would leverage the already available dynamic provision capabilities of the cloud. Alternatively, a detection mechanism based on bandwidth estimation tools is proposed. Traditional tools for detecting DoS attacks usually rely on the monitoring of signals such as CPU usage and available memory, neither of which is affected by this specific attack. Therefore, [56] introduces a novel mechanism that allows a host to accurately estimate how much bandwidth it has available. With that information in hand, it becomes possible to detect an imminent attack and trigger the appropriate avoidance countermeasures.

5 Accountability

Accountability has long been considered to be a property of trustworthy computer systems. An accountable system is capable of detecting misuse and exposing the responsible entity. It should be able to generate undeniable evidence of the entity's wrongful activity. Moreover, an innocent party in an accountable system should be capable of defending itself against any false accusations. Such properties are particularly valuable in environments where clients are running their tasks or services within the infrastructure owned or maintained by third parties which is the case for public clouds and distributed systems. Several types of systems in different domains can be used to exemplify the significance of accountability:

- In a shared file system outsourced to the cloud, users would like to be assured that the cloud service provider honestly propagates their updates on the data and that other users also do not tamper with the shared data in unauthorized ways.
- In collaborative and competitive systems (e.g., on-line games [40]) delegated to the cloud, each user would like to be able to know that other users indeed follow the agreed upon protocol.

- Users relying on (storage or computing) services offered by the cloud may want to be assured that the cloud service provider faithfully carries out the tasks as directed, utilizing the promised amount of resources.

To build an accountable system, several features should be taken into consideration:

- *Identity binding:* In order to undeniably link each action to the party that performed it, a binding mechanism [89] can be used. A widely utilized approach for achieving this consists of signing each action with the private key of the action originator. This will allow other entities who retain such records to present them to an auditing authority in order to prove their innocence or accuse the action originator of wrong doing.
- *Tamper-evident logs:* To investigate occurrence of malicious behavior of a certain entity, the auditing authority is normally presented with the history of that entity's past actions in the form of a log. One prominent property of this log is tamper-evidence. After making a commitment to the current state of the log, any subsequent attempt in tampering with the previous log entries invalidates the commitment. In [40], this feature is realized by building a hash chain from the log entries, where the current commitment is computed as a hash of the the previous commitment and the current log entry in the ordered log entry list. In other words, the ith commitment is computed as $C_i = H(e_i||C_{i-1})$, where here and in the rest of this work, $H(\cdot)$ denotes a collision-resistant hash function, e_i is the ith log entry, $||$ denotes concatenation, and C_0 can be set in any desired way.
- *Execution verification:* When a cloud service provider executes a task on behalf of a client and produces an outcome as the task's result, the client might be interested in verifying correctness of the result. For that purpose, Haeberlen et al. [40] suggest that the client could delegate the verification task to an auditing authority, and provide it with the log contents and records of non-deterministic events. In this context, non-deterministic events [15] are unpredictable events (e.g., hardware interrupts) that occur throughout the execution process which, together with deterministic events (e.g., messages sent or received), contribute to the final outcome of the task. With all such inputs, the auditing authority will re-execute the entire task using its trusted local resources to identify whether the outcome claimed in the log is producible. To save time for executing the complete log contents, the log can be split into several segments based on the snapshots taken during the task's execution process, so that the auditing authority can later perform spot checking of a few segments that begin and end at different snapshots. In addition to this spot-checking approach, in section 7.2 we discuss other available mechanisms for verifying integrity of the result of an outsourced task, which can be more efficient than this solution.

There are several publications that treat accountability in both the cloud [39, 40, 54, 80] and distributed systems [41, 89, 90] environments. They target either a server which does not fulfill the tasks delegated to it in a faithful manner, or peer entities who do not follow the agreed upon protocol, as being accountable. We next briefly review selected results in this domain.

Yumerefendi and Chase [90] proposed a network storage service that allows clients to access shared objects maintained by the server while achieving strong accountability. With this solution, any malicious behavior of either the server (e.g., incorrect execution of the clients' write requests) or a client (e.g., denying execution of unauthorized operations on shared data) can be captured. To assure clients' accountability, every request contains a digital signature that can be used to uniquely identify its sender and assures the integrity of the message. The collection of such signatures from all clients corresponding to their write requests constitutes the so-called action history. It is later used during an audit request on a particular object, during which the server presents a sequence of cached, signed write requests that prove to the auditor what the correct state of the object in question should be based on the actions of its clients. In that sense, after issuing write requests on a data object, a client cannot later deny his actions.

Regarding the server's accountability, the server is required to generate a new version of a data object for every write request on that object issued by a client and to organize all data objects in the form of a Merkle hash tree. In the standard Merkle hash tree structure, a leaf node corresponds to a hash value of an individual data object, and each internal node is assigned a value that corresponds to the hash of the concatenation of its children's values. For instance, a node containing two children with their respective values v_1 and v_2 will be assigned value $H(v_1||v_2)$. After building the Merkle hash tree in this manner, the server periodically publishes the root value of the Merkle hash tree to a trusted medium from which a client can independently retrieve it to verify the correctness of a requested data object as follows. When a client triggers a read request on a specific data object, the server sends the client the object itself, together with the sibling nodes that lie on the path from the leaf node (corresponding to the retrieved data object) to the root. The client then recomputes the root value based on the received information and compares it with the one retrieved from the trusted medium. If the two values match, the client is assured that the retrieved data object was indeed included in the server's data structure. To further ensure that the current content of the data object is the result of correct execution of the clients' prior write operations on the object, the client performs an audit on the object between two consecutive commitments by the server at, say, times t_1 and t_2. By verifying the history of the write operations performed on the object, the client is assured of the valid transition of the object's content during the time interval $[t_1, t_2]$. Lastly, to prevent the server from hiding the clients' write requests on the objects it stores, the clients themselves can communicate with each other and learn whether any client's operation is missing from the action history.

Jana and Shmatikov [45] introduced an approach for verifying whether interactive web applications hosted by the cloud were executed correctly. Their design is based on two key ideas: First, it provides a mapping mechanism for transforming every HTTP request received from users through its front-end to a read or write operation on the objects stored at the back-end storage. Second, to ensure accountability when dealing with multi-user web applications, a subset of users are selected to be the "witnesses" who keep the logs of their interactions with the cloud-hosted web applications. Later, these logs are sent to a trusted verifier for examination. Since the witnesses cannot be distinguished by the cloud service provider from other users, if the cloud provider's faults are random and independent, then it will be highly probable that the cloud provider will tamper with requests of the witnesses and will correspondingly be detected.

Xiao and Xiao [86] proposed P-Accountability – a metric for quantifying the degree of accountability in a networked system. It was developed to provide a trade-off between the level of accountability that can be achieved and the cost of building an accountable system. In particular, achieving perfect accountability typically involves a large amount of overhead. Furthermore, the noisy nature of network communication due to message loss and delay makes it difficult to link every event that occurred to the responsible entity. Accounting for these factors, the authors define the P-accountability metric as $\mathrm{P} = (\sum_{e \in E} I(e))/|E|$, where E is the event set, $e \in E$, and $I(e)$ returns 1 if e could be correctly traced back to the responsible entity, and 0 otherwise. Based on application demands, such as the amount of available resources and tolerable overhead, the system can specify different P values to achieve the desired level of accountability.

Recently, Sekar and Maniatis [67] proposed a systematic approach to account for the resources consumed by the cloud when executing clients' tasks. The problem the authors address in verifiable resource accounting is two-fold: (i) verifying whether the resources consumed for an outsourced task indeed matched the expenditures the client was charged for (i.e., "Did I use what I was charged for?"), and (ii) verifying whether the resource consumption can be justified based on the agreed upon policy (i.e., "Should I have used what I was charged for?"). Here, a policy determines a "reasonable" allocation of resources for a delegated workload. For instance, the client may describe an upper bound on the resources the server can dedicate to a task (which can be specified as a function of that task). The framework that the authors offer for answering these two questions relies on the use of trusted hardware. In more detail, for a given task, the server specifies the granularity of resource tracking and upon completion of the task the server generates a consumption report. The report describes the utilized resources with the specified granularity, and attestation of the hardware on which the task is run guarantees the integrity of the report. To verify the correctness of the billing, the client provides a third-party verifier with specification of the task and the server's report receives a Yes/No answer that indicates whether the server's resource usage was appropriate for the task.

6 Remote Storage Protection

Cloud storage refers to the increasingly prevalent on-line storage services hosted at the cloud. Thanks to the enormous storage capacity, high availability, and stable performance offered by the cloud, the applications resorting to cloud storage range from general file storage such as archiving and backup to web operating systems and databases. Using Amazon Simple Storage Service (S3) as an example, cloud storage providers tend to offer plain file-system like interfaces to the end users without exposing them to the complicated management of physical servers and facilities in which they reside. Moreover, because all storage servers of a cloud storage provider are centrally controlled through dynamic provisioning, the users are also relieved from the direct oversight of component reliability and outsider security threats.

While cloud computing offers appealing advantages, it also brings new security challenges. In particular, integrity of clients' outsourced data becomes a concern, which is caused by a lack of transparency and direct control in the cloud environment. While data confidentiality can be assured through traditional mechanisms such as encryption or secret sharing, and integrity of data transmission can be assured through data authentication, integrity of data stored in the cloud is significantly more difficult to verify. For that reason, the rest of this section deals with techniques for ensuring integrity of outsourced data.

After a client moves its data to the cloud, the client relinquishes its ultimate control over the data, which is now entirely managed by the cloud service provider. Thus, it is essential for the client to be able to verify that her data is still available at the cloud in its original form and is ready for retrieval when necessary. For instance, the client might want to make sure that her data has not been corrupted (deleted or modified) or moved to an off-line unavailable storage medium, that could be caused by either an attempt of a dishonest provider to save storage costs or by outages and security breaches within the cloud services themselves.

One possibility for assuring high availability of outsourced data is through simple replication, where a client stores its data with multiple server providers. This, however, results in unnecessarily high storage overhead and cost to the client. As an alternative, the client can store her data at a single provider, but either periodically retrieve and examine the entire data, or ask the server to periodically compute and send a one-way function of the stored data which the client then compares to a previously computed value that it expects. It is clear that these two approaches are also unsatisfactory as in the first one both the server and the user suffer from a substantial communication overhead and the second approach is prone to replay attacks by the server which can conceal the fact that the server tampered with the data. To mitigate these problems, a widely utilized approach [5, 6, 7, 13, 14, 18, 25, 27, 29, 48, 66, 69, 81, 84, 92] is to employ a challenge-response mechanism consisting of the following phases:

- *Challenge:* The client creates and sends to the cloud service provider (CSP) one or more unpredictable challenges, which are used to ensure that the CSP indeed retains the client's data at the time of auditing.
- *Response:* Upon receiving the challenge, the CSP derives its response from the challenge and the stored data, and sends the computed result back to the client.
- *Response verification:* After receiving a response from the CSP, the client compares it to the expected value that it previously pre-computed from the original data and its metadata. If the two values coincide, the client is assured that the data is stored at the CSP in its intact form.

A design objective of this framework is that a cloud service provider who is highly likely to store incomplete or incorrect data will be unable to respond to the challenges correctly, and thus the client will detect data integrity violations.

While security (i.e., guaranteeing that the server's misbehavior is detected) and efficiency (i.e., minimizing the client's and server's computation and communication) should be viewed as the main design criteria, in order to build a reliable and versatile protocol, it is desirable that several other properties are taken into consideration:

- *Unbounded number of audit queries:* A robust scheme should be able to support an unbounded number of audit protocol interactions to ensure that the server's misbehavior, which might occur at any time, will be detected. If a protocol only allows for a limited number of audit queries, upon reaching the limit, the client will have to retrieve the entire data from the server and re-initiate the scheme during which the data is stored at the server again. Clearly, this imposes additional overhead on clients which can be avoided.
- *Support for dynamic data operations:* In many circumstances, the data delegated to the cloud might need to be modified through the user's update requests (unlike backup archive files that are rarely modified). In this case, a protocol that works only for static operations (i.e., upload or read) is unlikely to be applicable to dynamic operations. The update operations can be insertions or deletions of new or existing blocks at arbitrary positions in the file, or modifications of existing blocks. With cloud storage, support for dynamic data operations can be of vital importance to both remote storage and database services.
- *Public verifiability:* Oftentimes, when performing integrity verification of outsourced data, the clients themselves are unreliable and are unable to consistently perform integrity checks. They may also lack necessary expertise for this task. For these reasons, it is desirable to develop mechanisms for public verification of data integrity by a trusted auditing server. The auditing server is a reliable and independent entity that challenges the cloud service provider on behalf of the clients and assures correctness of data storage, while not learning any information contained in the stored data. For improved efficiency, the auditing server could also perform batch auditing during which it simultaneously processes auditing requests from multiple users [79].

To ensure integrity of outsourced storage and address the above mentioned properties, many existing publications [5, 6, 7, 14, 18, 19, 27, 29, 43, 48, 66, 69, 78, 79, 81, 92, 94] proposed a number of techniques. In the following, we categorize prior work based on the design objectives and highlight cryptographic details and data structures used in these publications.

- *Basic scheme.* Juels and Kaliski [48] proposed one of the first solutions to this problem by implementing the basic functionality of proofs of retrievability (POR). In their sentinel-based scheme, the client first applies an error-correcting code to the original file, encrypts each file block, and then inserts a predefined number of randomly chosen sentinels into different positions in the file. Following a challenge-response protocol, the user challenges the cloud service provider by specifying a number of positions of sentinels, so that a distinct set of sentinels is selected for each challenge and their positions are unpredictable to the server. The user then requires the cloud service provider to return information about the blocks at the chosen positions and verifies their correctness by comparing the returned values with their locally stored copy. The scheme is considered secure if a server without the possession of the file can only succeed in producing correct responses with low probability. As can be expected, the security of the sentinel-based approach depends on the number of sentinels (defined by the security parameter) in each challenge, and the number may vary upon the user's demand on integrity assurance (e.g., 90% of user's data should be kept intact). Given a sufficient number of sentinels in a challenge, if the server tampers with a large portion of the file, it will inevitably corrupt some of the sentinels that are being checked and the scheme will correspondingly detect the integrity violation with high probability. Since the number of embedded sentinels is independent of the length of the file stored at the server, the communication and storage costs for verifying the file's integrity remain a small constant. Another notable feature of the scheme is that a small corruption of the file that may not be detected using the spot-checking mechanism can still be recovered using the error-correction code embedded into the file. In spite of this basic functionality, the scheme has several disadvantages: it does not permit an unbounded number of audit queries, lacks support for dynamic operations, and does not allow for public verifiability.

- *Schemes with support for public verifiability.* The solutions in [5, 29, 69, 14, 81, 79, 92] support both public verifiability and an unlimited number of auditing interactions. A common approach for achieving these two properties is to utilize the so-called homomorphic authenticators [5, 69, 81, 79]. A homomorphic authenticator is an unforgeable metadata derived from a file block, with the property that several homomorphic authenticators can be securely aggregated. When a client would like to verify the integrity of a number of file blocks, it is sufficient for the client to verify only the aggregated homomorphic authenticator that corresponds to a linear combination of the individual blocks. Since the only information necessary for carrying out verification of correct data storage using a challenge-response protocol is the client's public key and the challenge data, the auditing process can be delegated to a third party who is equipped with sufficient computing and storage resources.

- *Schemes with support for dynamic operations.* Publications [6, 29, 43, 78, 81, 94] provide schemes that support dynamic updates on the outsourced data. The work [6, 78] implemented this functionality by using carefully designed sentinels (unlike [48], where the sentinels are assigned random values) and outsourcing them together with the client's data. Another line of research [29, 81, 43, 94] that supports dynamic operations utilizes specialized data structures such as Merkle hash trees or skip lists to organize information about the data blocks outsourced to a cloud service provider. When a Merkle hash tree structure is used for this purpose, each leaf node corresponds to the hash of an individual data block in the file being outsourced, and each internal node is assigned a value that corresponds to the hash of the concatenation of its children's values. The client only needs to store the root value of the tree. When the client retrieves a data block from the cloud server, the server sends the values of the sibling nodes lying on the path from the data block to the root. Having this information, the client verifies the correctness of the retrieved data block by recomputing the root value and compares it to the one that it locally stores.

7 Privacy and Integrity of Outsourced Computation

This section treats security issues that arise in the context of outsourced computation, and in particular, we address the broad topics of privacy and integrity protection of outsourced computation.

7.1 Privacy Protection of Outsourced Computation

As companies, individuals, and even governments increasingly outsource computation to the cloud, they knowingly relinquish the ultimate control over their data and the computation. If the computation involves proprietary, private, or otherwise sensitive data, cloud users might wish to protect the privacy of the data being used in the process, particularly when the infrastructure provider cannot be fully trusted. Although the concept of cloud computing is relatively new, privacy protection of outsourced computation has been studied for many years and most of the developments in this research area can be directly applied to the cloud architecture. There are currently a number of solutions that allow different types of operations (i.e., matrix multiplication, modular exponentiation, etc.) or more general functionality to be performed by the cloud provider on data that has been preprocessed by the user to ensure the safety of sensitive information. In what follows, we provide a more detailed description of some of these approaches.

All existing solutions in the literature can be grouped into the following categories: (1) schemes where the client outsources its computation to a single server, (2) solutions where the client uses multiple (normally two) servers for its computational task, but there is no interaction between the servers, and (3) protocols where a task is outsourced to multiple (two or more) servers that jointly carry out the computation. The solutions from the second category normally utilize two servers to ensure correctness of the result in the presence of faulty servers rather than in a more fundamental way. We start our description with the literature where the computation is outsourced to a single server followed by multi-server solutions.

Homomorphic encryption schemes have special properties that allow computation on ciphertexts to translate to meaningful operations on the plaintexts that they encrypt, and can be used in the context of secure outsourcing to a single server.

While an overview of homomorphic schemes is beyond the scope of this chapter, we mention that the existing techniques have limitations either in the class of functions that are able to evaluate on the encrypted data or in their performance.

In recent years, there has been an increasing interest in privately outsourcing complex (scientific) operations. Atallah et al. [4] proposed techniques for securely outsourcing several computations, with applications ranging from linear algebra to string pattern matching. As no single proposed mechanism was broad enough to be applied to a large variety of scientific problems, that work resorts to a number of different disguise techniques that are tuned to specific situations to enable secure outsourcing of most scientific computations at low cost. In [2], the protocols for outsourcing linear algebra operations are further improved to work with a single (or multiple) external servers that, unlike in previous protocols, no longer require expensive cryptographic operations. Another attractive property of this protocol is that it allows for a continuous (and safe) chain of outsourcing. For instance, by using this technique, a client can outsource a large matrix multiplication to the cloud which in its turn, can also delegate part of that computation to another remote server. In each step the client outsourcing the computation does only linear in the size of the matrix work, and can detect any attempt by the server to corrupt the answer.

As we next proceed with techniques for secure outsourcing to multiple servers, where the servers jointly carry out the computation, we note that many techniques from secure multi-party computation (SMC) can be used in this setting. Their overview, however, is beyond the scope of this chapter. We only mention that the client often can split its input between the computational servers, who can then proceed with the secure computation and return the result to the client.

We also mention works that deal with important applications. Atallah et al. [3] describe a secure outsourcing technique via which a client that does not have the necessary computational power can outsource large sequence comparisons (e.g., DNA matching) to two remote computing nodes, such that the result of the comparison and the two sequences being compared are known only by the client. This problem was revisited in [11], where a solution of improved performance was put forth. Also, the work [10] describes the means by which oblivious evaluation of a finite automaton on a private input can be securely outsourced to two or more servers. Though there are many operations that can be approached by the given technique such as searching for a regular expression in a database, the paper focuses on DNA searching and gives optimizations for that specific setting.

Finally, a couple of recent publications propose privacy-preserving architectures for MapReduce-based [26] computation. The work by Roy et al. [64] recognized the need for an automated system that could be used by clients with no knowledge of security to enforce personalized access control policies to their data. With that premise, the authors designed and implemented Airavat, which incorporated the mandatory access control features of SELinux to MapReduce, and also provided measures to avoid the leakage of information through the output of computations. Zhang et al. [93] considered a slightly different scenario, in which a given entity wishes to outsource a large computation where only a portion of the data is sensitive and needs to be processed by a private cloud, whereas the remaining information can be safely outsourced to a public cloud. Also building upon MapReduce, the authors proposed Sedic, a generic secure computing framework for hybrid clouds. Sedic allows the end user to continue to use the legacy data-processing code with the added functionality of having the data being automatically partitioned between private and public and adequately outsourced according to that labeling. Several techniques are also suggested in order to maximize the amount of computation done in the public cloud side, which increases the overall performance.

7.2 Outsourced Verifiable Computation

As mentioned earlier, a prominent advantage of the cloud paradigm is providing computing resources to resource-constrained users who are now able to outsource their computation-intensive tasks to the cloud with abundant processing, storage, and bandwidth resources. With this architecture, however, the clients no longer have direct control over the outsourced data and computation, and there is a lack of transparency in the current cloud services. The cloud provider can be incentivized to delete rarely accessed data or skip some of the computations to conserve resources (for financial or other reasons), which is especially true for volunteer-based computational clouds. Furthermore, unintentional data or computation corruption might also take place for a variety of reasons including malware, security break-ins, etc. From that perspective, it is important for the clients to be able to verify the correctness of the result of the outsourced task. Furthermore, the verification mechanism should be such that it does not require the client to perform a computation comparable in size to the outsourced task itself. Techniques that enable verification of correctness of the result for outsourced computation are treated in this section.

To formulate the problem, a client delegates the task of computing $f(x)$ for a set of inputs $X = \{x_1, x_2, \ldots, x_n\}$ to a cloud service provider. A dishonest cloud provider evaluates the function on a subset of the inputs $X' \subset X$, and for the rest of the inputs it returns the result of evaluation of $f'(\cdot)$, where $f'(\cdot)$ is cheaper to compute than $f(\cdot)$. This type of behavior is termed "lazy." Alternatively, a part of the result might be corrupt, but normally it is assumed that the cloud server will not maliciously modify the result (by performing work equivalent to or exceeding that of computing the result correctly). After receiving the results, the client would like to verify the output in an efficient manner and detect any deviation from the prescribed computation by the cloud provider.

Prior literature contains a number of techniques [2, 9, 76, 28, 35, 37, 50, 53, 55, 73, 82, 85] for efficiently verifying the result of an outsourced computing task. We categorize them into two groups based on the type of computation to which the techniques are applied.

7.2.1 Function-Independent Computation

The techniques in this category can be applied to general types of computation. The common approach of these techniques consists of assigning the extra workload to the server that is identical to computation being outsourced, and later analyzing the computation results of both the original task and additional workload to determine any deviation from the prescribed computation by the service provider. This extra workload can take the form of either insertion of so-called ringers [35, 28, 50] or duplicate task execution [36, 73, 37, 53, 82, 85]. We provide additional descriptions of some of the typical mechanisms.

Insertion of ringers was among the first proposals for addressing this problem and is now commonly used. Ringer r [35] in this context, is a value chosen by the client in the domain of the function f that it outsources to a server, and the client is assumed to be equipped with the knowledge of $f(r)$.[1] Furthermore, the computational server should be unable to distinguish ringers from the inputs of the original task. As a result, if the cloud provider corrupts a non-negligible portion of the computation, then with high probability, it will tamper with the computation of the ringers, which will be detected by comparing the returned results with the client's pre-computed $f(r)$. Golle and Mironov [35] applied the ringer-based technique to search for rare events for the computation that involved inverting a one-way function h (i.e., the client outsources the computation of h^{-1}).

Szajda et al. [73] used an alternative technique that consists of duplicate task execution for a number of sequential workloads that can be subdivided into independent sub-tasks. The approach assumes that multiple servers which do not collude with each other, execute a task on behalf of a client. To guarantee redundancy, it is also necessary that the number of servers is greater than the number of workloads. Under these assumptions, the proposed mechanism guarantees that the result of the outsourced computation is correct with high probability. By the law of large numbers, the correct result for each sub-task is highly likely to be the result returned by the majority of the servers; the correct computational task can then be ascertained.

Canetti et al. [17] recently proposed a general strategy for verifiable task outsourcing where a client outsources the computation to multiple servers and is guaranteed to obtain the correct result as long as at least one server is honest. The approach proposed by the authors works as follows: Given the number of instructions that compose the computation task, the client uses a binary search to determine the exact location where the intermediate states of two (or more) executions begin to disagree. Upon locating the inconsistency, the client is able to determine which execution was correct by carrying out one step of the computation himself. One requirement of this scheme is that execution of the delegated task must be fully deterministic as it will be non-trivial to distinguish an outcome that is incorrect as a result of a stochastic event from an outcome that is incorrect as a result of a malicious event. In addition, Gennaro et al. [32] and Chung et al. [23] recently proposed solutions that explicitly combine privacy with integrity verification of outsourced

[1] We note that our definition of a ringer is different from the original definition in [35] to ensure that it can be applied to any functionality f that the client wishes to outsource.

computation and are applicable to any computable function. Gennaro et al. [32] describe a non-interactive verifiable computation outsourcing scheme that combines Yao's garbled circuit evaluation [88] with a fully homomorphic encryption scheme (e.g., Gentry's proposal [34]). Because evaluating a garbled circuit on two different inputs cannot provide adequate security, the authors propose that the circuit is evaluated on encrypted inputs. That is, after creating a garbled Boolean circuit for the desired functionality, the client stores it at the server. When the client desires to evaluate the circuit on an input, it encrypts the labels associated with the circuit's input bits using the public key of a fully homomorphic encryption scheme and sends the encrypted labels to the server. The cloud server evaluates the function on the encrypted data using the homomorphic properties of the encryption scheme. After the server evaluates the entire function and returns encryptions of the output wire labels to the client, the client decrypts the ciphertexts, recovers the labels, and interprets the result.

Setty et al. [68] recently proposed another type of general strategy for verifiable outsourced computation based on an argument system with probabilistically checkable proofs (PCP). In an argument system, a computationally bounded prover tries to convince the verifier that the delegated computation was computed correctly, and with PCP, the verifier is able to guarantee correctness of the outsourced computation with high probability by probing only a constant number of positions within the proof. The solution builds on the work of Kilian [51, 52] in which the prover is asked to make a commitment to his computation that should be consistent with the queries that will be issued afterwards by the verifier. Setty et al. constructed a scheme that aims to work for a general computation with practical performance by incorporating several efficiency improvements to the naive implementation of [44]. The improvements include using arithmetic circuits with compact gates instead of Boolean circuits to represent the function, and amortizing the verifier's query costs through batching.

7.2.2 Function-Specific Computation

The techniques in this category exploit knowledge of the domain function of the computation to efficiently validate the results of an outsourced task. Algebraic computations [9, 2] and linear programming [76] are typical computation domains that have been studied in some depth. Also, many of the available techniques combine privacy protection of the data used throughout the outsourced computation with verification of the result of the computation.

Wang et al. [76] proposed a result verification mechanism for linear programming (LP) in cloud computing. In this approach, outsourcing the computation of linear programming means sending all parameters of the LP problem to the cloud service provider in obfuscated form to preserve privacy. To verify the returned solution, [76] utilizes the duality theorem to construct a secondary LP problem and derive necessary and sufficient conditions for an outcome to be valid. By outsourcing these two LP problems to the cloud service provider and later comparing the returned results, the client can verify the correctness of the original LP problem. In a similar vein, Benjamin et al. [9] and Atallah et al. [2] proposed solutions that deal with secure and private outsourcing of linear algebra computations (matrix multiplication).

Blanton et al. [12] recently studied the problem of verifiable outsourcing of large-scale biometric computations. The authors treat the computation of all-pairs distances between two sets of biometric data that produces a distance matrix, as well as the corresponding distribution computation that calculates the frequency of each distinct distance appearing in the distance matrix. The work focuses on three distance metrics, namely the Hamming distance, the Euclidean distance, and the set intersection cardinality. In this solution, to verify the integrity of both the all-pairs and the distance distribution computation, the client inserts fake biometric items into the computation, which are indistinguishable from the real biometric items in the datasets. The fake biometrics are carefully designed so that the range of distances between two real biometric items is separated from the range of distances between real and fake biometrics. By comparing several values returned by the server with their expected values that the client precomputes, the client achieves the desired level of assurance that the remaining values were computed honestly.

In addition to the above literature, [8] provides verifiable computation schemes for evaluation of high degree polynomials, which can also be applied to verifiable keyword searches and proofs of retrievability. Another work [59] develops a mechanism for verifying the result of outsourced set operations using authenticated data structures, with applications ranging from efficient verification of keyword search to database queries. Finally, Wang et al. [77] proposed a solution for securely outsourcing large-scale systems of linear equations. The solution is based on an iterative method and incurs only $O(n)$ local computations on the customer's side for a linear system with $n \times n$ coefficient matrix.

8 Conclusions

Cloud security has emerged as an important topic both in the research community and in practice due to the recent rapid growth in the availability and popularity of cloud computing and storage providers. It is often cited as a top reason for hesitance in adopting cloud computing services by companies and is a difficult problem that re-surfaces old security issues and brings new ones. In the attempt to bring wider attention to the topic, this work provides the first comprehensive treatment of the area by providing a literature overview of cloud security as well as secure remote storage and computation. The topics covered in this survey include:

- authentication and authorization attacks and countermeasures
- virtualization-related security issues
- denial of service attacks
- solutions that address accountability
- schemes for integrity verification of remote storage

- techniques for privacy and integrity protection of outsourced computation.

As the field matures, we expect to see techniques for more complex and interoperable cloud interactions such as multi-user access to remote storage and tasks' configuration automation for use with cloud providers, which will need to be treated in the security literature as well.

Acknowledgements This work was supported in part by the Air Force Office of Scientific Research (AFOSR) under Grant No. AFOSR-FA9550-09-1-0223.

References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Zaharia, M.: Above the clouds: A Berkeley view of cloud computing. Tech. Rep. UCB/EECS-2009-28, University of California at Berkeley (2009)
2. Atallah, M., Frikken, K.: Securely outsourcing linear algebra computations. In: Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ASIACCS'10, pp. 48–59. ACM, New York, NY, USA (2010). DOI 10.1145/1755688.1755695
3. Atallah, M., Li, J.: Secure outsourcing of sequence comparisons. International Journal of Information Security **4**(4), 277–287 (2005)
4. Atallah, M., Pantazopoulos, K., Rice, J., Spafford, E.: Secure outsourcing of scientific computations. Advances in Computers **54**, 216–272 (2001)
5. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., Song, D.: Provable data possession at untrusted stores. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, CCS'07, pp. 598–609. ACM, New York, NY, USA (2007). DOI 10.1145/1315245.1315318
6. Ateniese, G., Di Pietro, R., Mancini, L., Tsudik, G.: Scalable and efficient provable data possession. In: Proceedings of the 4th International Conference on Security and Privacy in Communication Networks, SecureComm'08, pp. 9:1–9:10. ACM, New York, NY, USA (2008). DOI 10.1145/1460877.1460889
7. Ateniese, G., Kamara, S., Katz, J.: Proofs of storage from homomorphic identification protocols. In: Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology, ASIACRYPT'09, pp. 319–333. Springer-Verlag, Berlin, Heidelberg (2009). DOI 10.1007/978-3-642-10366-7_19
8. Benabbas, S., Gennaro, R., Vahlis, Y.: Verifiable delegation of computation over large datasets. In: Proceedings of the 31st Annual Conference on Advances in Cryptology, CRYPTO'11, pp. 111–131. Springer-Verlag, Berlin, Heidelberg (2011)
9. Benjamin, D., Atallah, M.: Private and cheating-free outsourcing of algebraic computations. In: Proceedings of the 6th Annual Conference on Privacy, Security and Trust, PST'08, pp. 240–245. IEEE Computer Society, Washington, DC, USA (2008). DOI 10.1109/PST.2008.12
10. Blanton, M., Aliasgari, M.: Secure outsourcing of DNA searching via finite automata. In: Proceedings of the Annual IFIP WG 11.3 Working Conference on Data and Applications Security, DBSec'10, pp. 49–64. Springer-Verlag, Berlin, Heidelberg (2010). DOI 10.1007/978-3-642-13739-6_4
11. Blanton, M., Atallah, M., Frikken, K., Malluhi, Q.: Secure and efficient outsourcing of sequence comparisons. In: Proceedings of the European Symposium on Research in Computer Security, ESORICS'12, pp. 505–522. Springer-Verlag, Berlin, Heidelberg (2012). DOI 10.1007/978-3-642-33167-1_29

12. Blanton, M., Zhang, Y., Frikken, K.: Secure and verifiable outsourcing of large-scale biometric computations. In: Proceedings of the IEEE International Conference on Information Privacy, Security, Risk and Trust, PASSAT'11, pp. 1185–1191 (2011). DOI 10.1109/PASSAT/SocialCom.2011.13
13. Bowers, K., Juels, A., Oprea, A.: HAIL: A high-availability and integrity layer for cloud storage. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, CCS'09, pp. 187–198. ACM, New York, NY, USA (2009). DOI 10.1145/1653662.1653686
14. Bowers, K., Juels, A., Oprea, A.: Proofs of retrievability: Theory and Implementation. In: Proceedings of the ACM Workshop on Cloud Computing Security, CCSW'09, pp. 43–54. ACM, New York, NY, USA (2009). DOI 10.1145/1655008.1655015
15. Bressoud, T., Schneider, F.: Hypervisor-based fault tolerance. In: Proceedings of the 15th ACM Symposium on Operating Systems Principles, SOSP'95, pp. 1–11. ACM, New York, NY, USA (1995). DOI 10.1145/224056.224058
16. Bugiel, S., Nürnberger, S., Pöppelmann, T., Sadeghi, A.R., Schneider, T.: AmazonIA: When elasticity snaps back. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS'11, pp. 389–400. ACM, New York, NY, USA (2011). DOI 10.1145/2046707.2046753
17. Canetti, R., Riva, B., Rothblum, G.: Practical delegation of computation using multiple servers. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS'11, pp. 445–454. ACM, New York, NY, USA (2011). DOI 10.1145/2046707.2046759
18. Chang, E., Xu, J.: Remote integrity check with dishonest storage server. In: Proceedings of the 13th European Symposium on Research in Computer Security, ESORICS'08, pp. 223–237. Springer-Verlag, Berlin, Heidelberg (2008). DOI 10.1007/978-3-540-88313-5_15
19. Chen, B., Curtmola, R., Ateniese, G., Burns, R.: Remote data checking for network coding-based distributed storage systems. In: Proceedings of the ACM Cloud Computing Security Workshop, CCSW'10, pp. 31–42. ACM, New York, NY, USA (2010). DOI 10.1145/1866835.1866842
20. Chen, Y., Paxson, V., Katz, R.: What's new about cloud computing security? Tech. Rep. UCB/EECS-2010-5, Electrical Engineering and Computer Sciences, University of California at Berkeley (2010). URL `http://bit.ly/ThcYr3`
21. Chow, R., Golle, P., Jakobsson, M., Shi, E., Staddon, J., Masuoka, R., Molina, J.: Controlling data in the cloud: Outsourcing computation without outsourcing control. In: Proceedings of the ACM Cloud Computing Security Workshop, CCSW'09, pp. 85–90. ACM, New York, NY, USA (2009). DOI 10.1145/1655008.1655020
22. Christodorescu, M., Sailer, R., Schales, D., Sgandurra, D., Zamboni, D.: Cloud security is not (just) virtualization security. In: Proceedings of the ACM Cloud Computing Security Workshop, CCSW'09, pp. 97–102. ACM, New York, NY, USA (2009). DOI 10.1145/1655008.1655022
23. Chung, K.M., Kalai, Y., Vadhan, S.: Improved delegation of computation using fully homomorphic encryption. In: Proceedings of the 30th Annual Conference on Advances in Cryptology, CRYPTO'10, pp. 483–501. Springer-Verlag, Berlin, Heidelberg (2010)
24. Clark, D., Wilson, D.: A comparison of commercial and military computer security policies. In: Proceedings of the IEEE Symposium on Security and Privacy, pp. 184–194 (1987). DOI 10.1109/SP.1987.10001
25. Curtmola, R., Khan, O., Burns, R., Ateniese, G.: MR. PDP: Multiple-replica provable data possession. In: Proceedings of the 28th International Conference on Distributed Computing Systems, ICDCS'08, pp. 411–420. IEEE Computer Society, Washington, DC, USA (2008). DOI 10.1109/ICDCS.2008.68
26. Dean, J., Ghemawat, S.: MapReduce: Simplified data processing on large clusters. Communications of the ACM **51**(1), 107–113 (2008). DOI 10.1145/1327452.1327492
27. Dodis, Y., Dadhan, S., Wichs, D.: Proofs of retrievability via hardness amplification. In: Proceedings of the 6th Theory of Cryptography Conference, TCC'09, pp. 109–127. Springer-Verlag, Berlin, Heidelberg (2009). DOI 10.1007/978-3-642-00457-5_8

28. Du, W., Goodrich, M.: Searching for high-value rare events with uncheatable grid computing. In: Proceedings of the 3rd International Conference on Applied Cryptography and Network Security, ACNS'05, pp. 122–137. Springer-Verlag, Berlin, Heidelberg (2005). DOI 10.1007/11496137_9
29. Erway, C., Kupcu, A., Papamanthou, C., Tamassia, R.: Dynamic provable data possession. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, CCS '09, pp. 213–222. ACM, New York, NY, USA (2009). DOI 10.1145/1653662.1653688
30. Eucalyptus Systems, Inc.: Eucalyptus: Open source private and hybrid clouds. http://www.eucalyptus.com/ (2012)
31. Garfinkel, T., Rosenblum, M.: A virtual machine introspection based architecture for intrusion detection. In: Proceedings of Network and Distributed Systems Security Symposium, NDSS'03, pp. 191–206 (2003)
32. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: Proceedings of the 30th Annual Conference on Advances in Cryptology, CRYPTO'10, pp. 465–482. Springer-Verlag, Berlin, Heidelberg (2010)
33. Gens, F.: IT cloud services user survey, pt. 2: Top benefits & challenges. http://bit.ly/oUCbY (2008)
34. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC'09, pp. 169–178. ACM, New York, NY, USA (2009). DOI 10.1145/1536414.1536440
35. Golle, P., Mironov, I.: Uncheatable distributed computations. In: Proceedings of the 2001 Conference on Topics in Cryptology: The Cryptographer's Track at the RSA Conference, CT-RSA'01, pp. 425–440. Springer-Verlag, London, UK, UK (2001)
36. Golle, P., Stubblebine, S.: Secure distributed computing in a commercial environment. In: Proceedings of the 5th International Conference on Financial Cryptography, FC'01, pp. 289–304. Springer-Verlag, London, UK, UK (2001)
37. Goodrich, M.: Pipelined algorithms to detect cheating in long-term grid computations. Theoretical Computer Science **408**(2–3), 199–207 (2008). DOI 10.1016/j.tcs.2008.08.008
38. Gruschka, N., Iacono, L.: Vulnerable cloud: SOAP message security validation revisited. In: Proceedings of the IEEE International Conference on Web Services, ICWS'09, pp. 625–631. IEEE Computer Society, Washington, DC, USA (2009). DOI 10.1109/ICWS.2009.70
39. Haeberlen, A.: A case for the accountable cloud. ACM SIGOPS Operating System Review **44**(2), 52–57 (2010). DOI 10.1145/1773912.1773926
40. Haeberlen, A., Aditya, P., Rodrigues, R., Druschel, R.: Accountable virtual machines. In: Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, OSDI'10, pp. 119–134. USENIX Association, Berkeley, CA, USA (2010)
41. Haeberlen, A., Kouznetsov, P., Druschel, P.: PeerReview: Practical accountability for distributed systems. In: Proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles, SOSP'07, pp. 175–188. ACM, New York, NY, USA (2007). DOI 10.1145/1294261.1294279
42. Hamlen, K., Kantarcioglu, M., Khan, L., Thuraisingham, B.: Security issues for cloud computing. International Journal of Information Security and Privacy (IJISP) **4**(2), 36–48 (2010)
43. Heitzmann, A., Palazzi, B., Papamanthou, C., Tamassia, R.: Efficient integrity checking of untrusted network storage. In: Proceedings of the 4th ACM International Workshop on Storage Security and Survivability, StorageSS'08, pp. 43–54. ACM, New York, NY, USA (2008). DOI 10.1145/1456469.1456479
44. Ishai, Y., Kushilevitz, E., Ostrovsky, R.: Efficient arguments without short PCPs. In: Proceedings of the 22nd Annual IEEE Conference on Computational Complexity, CCC'07, pp. 278–291. IEEE Computer Society, Washington, DC, USA (2007). DOI 10.1109/CCC.2007.10
45. Jana, S., Shmatikov, V.: EVE: Verifying correct execution of cloud-hosted web applications. In: Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing, HotCloud'11. USENIX Association, Berkeley, CA, USA (2010)
46. Jensen, M., Gruschka, N., Herkenhoner, R.: A survey of attacks on web services. Computer Science – Research and Development **24**(4), 185–197 (2009)

47. Jensen, M., Schwenk, J., Gruschka, N., Iacono, L.: On technical security issues in cloud computing. In: Proceedings of the IEEE International Conference on Cloud Computing, CLOUD'09, pp. 109–116. IEEE Computer Society, Washington, DC, USA (2009). DOI 10.1109/CLOUD.2009.60
48. Juels, A., Kaliski, B.: PORs: Proofs of retrievability for large files. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, CCS'07, pp. 584–597. ACM, New York, NY, USA (2007). DOI 10.1145/1315245.1315317
49. Kandukuri, B., Paturi V, R., Rakshit, A.: Cloud security issues. In: Proceedings of the IEEE International Conference on Services Computing, SCC'09, pp. 517–520. IEEE Computer Society, Washington, DC, USA (2009). DOI 10.1109/SCC.2009.84
50. Karame, G., Strasser, M., Capkun, S.: Secure remote execution of sequential computations. In: Proceedings of the 11th International Conference on Information and Communications Security, ICICS'09, pp. 181–197. Springer-Verlag, Berlin, Heidelberg (2009). DOI 10.1007/978-3-642-11145-7_15
51. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: Proceedings of the 24th Annual ACM Symposium on Theory of Computing, STOC'92, pp. 723–732. ACM, New York, NY, USA (1992). DOI 10.1145/129712.129782
52. Kilian, J.: Improved efficient arguments. In: Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology, CRYPTO'95, pp. 311–324. Springer-Verlag, London, UK (1995)
53. Kim, H., Gil, J., Hwang, C., Yu, H., Joung, S.: Agent-based autonomous result verification mechanism in desktop grid systems. In: Proceedings of the International Workshop on Agents and Peer-to-Peer Computing, AP2PC'07, pp. 72–84. Springer-Verlag, Berlin, Heidelberg (2007). DOI 10.1007/978-3-642-11368-0_6
54. Ko, R., Jagadpramana, P., Mowbray, M., Pearsoni, S., Kirchberg, M., Liang, Q., Lee, B.: TrustCloud: A framework for accountability and trust in cloud computing. In: Proceedings of the IEEE World Congress on Services, SERVICES'11, pp. 584–588. IEEE Computer Society, Washington, DC, USA (2011). DOI 10.1109/SERVICES.2011.91
55. Kuhn, M., Schmid, S., Watterhofer, R.: Distributed asymmetric verification in computational grids. In: Proceedings of the IEEE International Symposium on Parallel and Distributed Processing, IPDPS'08, pp. 1–10 (2008). DOI 10.1109/IPDPS.2008.4536244
56. Liu, H.: A new form of DOS attack in a cloud and its avoidance mechanism. In: Proceedings of the ACM Cloud Computing Security Workshop, CCSW'10, pp. 65–76. ACM, New York, NY, USA (2010). DOI 10.1145/1866835.1866849
57. McIntosh, M., Austel, P.: XML signature element wrapping attacks and countermeasures. In: Proceedings of the Workshop on Secure Web Services, SWS'05, pp. 20–27. ACM, New York, NY, USA (2005). DOI 10.1145/1103022.1103026
58. Mell, P., Grance, T.: The NIST definition of cloud computing. National Institute of Standards and Technology (NIST) **53**(6) (2009)
59. Papamanthou, C., Tamassia, R., Triandopoulos, N.: Optimal verification of operations on dynamic sets. In: Proceedings of the 31st Annual Conference on Advances in Cryptology, CRYPTO'11, pp. 91–110. Springer-Verlag, Berlin, Heidelberg (2011)
60. Pearson, S., Benameur, A.: Privacy, security and trust issues arising from cloud computing. In: Proceedings of the International Workshop on Cloud Privacy, Security, Risk, and Trust, CLOUDCOM'10, pp. 693–702. IEEE Computer Society, Washington, DC, USA (2010). DOI 10.1109/CloudCom.2010.66
61. Pfoh, J., Schneider, C., Eckert, C.: A formal model for virtual machine introspection. In: Proceedings of the 1st ACM Workshop on Virtual Machine Security, VMSec'09, pp. 1–10. ACM, New York, NY, USA (2009). DOI 10.1145/1655148.1655150
62. Popović, K., Hocenski, Z.: Cloud computing security issues and challenges. In: Proceedings of the 33rd International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO'10, pp. 344–349 (2010)
63. Ristenpart, T., Tromer, E., Shacham, H., Savage, S.: Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. In: Proceedings of the 16th ACM Con-

ference on Computer and Communications Security, CCS'09, pp. 199–212. ACM, New York, NY, USA (2009). DOI 10.1145/1653662.1653687
64. Roy, I., Setty, S., Kilzer, A., Shmatikov, V., Witchel, E.: Airavat: Security and privacy for MapReduce. In: Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, NSDI'10, pp. 297–312. USENIX Association, Berkeley, CA, USA (2010)
65. Schiffman, J., Moyer, T., Shal, C., Jaeger, T., McDaniel, P.: Justifying integrity using a virtual machine verifier. In: Proceedings of the Computer Security Applications Conference, ACSAC'09, pp. 83–92. IEEE Computer Society, Washington, DC, USA (2009). DOI 10.1109/ACSAC.2009.18
66. Sebe, F., Domingo-Ferrer, J., Martinez-Belleste, A., Deswarte, Y., Quisquater, J.J.: Efficient remote data possession checking in critical information infrastructures. IEEE Transactions on Knowledge and Data Engineering **20**(8), 1034–1038 (2008). DOI 10.1109/TKDE.2007.190647
67. Sekar, V., Maniatis, P.: Verifiable resource accounting for cloud computing services. In: Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW'11, pp. 21–26. ACM, New York, NY, USA (2011). DOI 10.1145/2046660.2046666
68. Setty, S., McPherson, R., Blumberg, A., Walfish, M.: Making argument systems for outsourced computation practical (sometimes). In: Proceedings of the Network and Distributed Systems Security Symposium, NDSS'12 (2012)
69. Shacham, H., Waters, B.: Compact proofs of retrievability. In: Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology, ASIACRYPT '08, pp. 90–107. Springer-Verlag, Berlin, Heidelberg (2008). DOI 10.1007/978-3-540-89255-7_7
70. Somorovsky, J., Heiderich, M., Jensen, M., Schwenk, J., Gruschka, N., Iacono, L.: All your clouds are belong to us: Security analysis of cloud management interfaces. In: Proceedings of the ACM Cloud Computing Security Workshop, CCSW'11, pp. 3–14. ACM, New York, NY, USA (2011). DOI 10.1145/2046660.2046664
71. Stamm, S., Ramzan, Z., Jakobsson, M.: Drive-by pharming. In: Proceedings of the International Conference on Information and Communications Security, ICICS'07, pp. 495–506. Springer-Verlag, Berlin, Heidelberg (2007)
72. Symantec Corporation: 2011 state of cloud survey. `http://bit.ly/naMLK3` (2011)
73. Szajda, D., Lawson, B., Owen, J.: Hardening functions for large scale distributed computations. In: Proceedings of the IEEE Symposium on Security and Privacy, SP'03, pp. 216–225. IEEE Computer Society, Washington, DC, USA (2003)
74. Szefer, J., Keller, E., Lee, R., Rexford, J.: Eliminating the hypervisor attack surface for a more secure cloud. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS'11, pp. 401–412. ACM, New York, NY, USA (2011). DOI 10.1145/2046707.2046754
75. Vaquero, L., Rodero-Merino, L., Morn, D.: Locking the sky: A survey on IaaS cloud security. Computing **91**(1), 93–118 (2011). DOI 10.1007/s00607-010-0140-x
76. Wang, C., Ren, K., Wang, J.: Secure and practical outsourcing of linear programming in cloud computing. In: Proceedings of the IEEE International Conference on Computer Communications, INFOCOM'11, pp. 820–828 (2011). DOI 10.1109/INFCOM.2011.5935305
77. Wang, C., Ren, K., Wang, J., Urs, K.: Harnessing the cloud for securely solving large-scale systems of linear equations. In: Proceedings of the 31st International Conference on Distributed Computing Systems, ICDCS'11, pp. 549–558. IEEE Computer Society, Washington, DC, USA (2011). DOI 10.1109/ICDCS.2011.41
78. Wang, C., Wang, Q., Ren, K., Lou, W.: Ensuring data storage security in cloud computing. In: Proceedings of the International Workshop on Quality of Service (2009). DOI 10.1109/IWQoS.2009.5201385
79. Wang, C., Wang, Q., Ren, K., Lou, W.: Privacy-preserving public auditing for data storage security in cloud computing. In: Proceedings of the 29th IEEE International Conference on Computer Communications, INFOCOM'10, pp. 525–533. IEEE Press, Piscataway, NJ, USA (2010)

80. Wang, C., Zhou, Y.: A collaborative monitoring mechanism for making a multitenant platform accountable. In: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, HotCloud'10. USENIX Association, Berkeley, CA, USA (2010)
81. Wang, Q., Wang, C., Li, J., Ren, K., Lou, W.: Enabling public verifiability and data dynamics for storage security in cloud computing. In: Proceedings of the 14th European Symposium on Research in Computer Security, ESORICS'09, pp. 355–370. Springer-Verlag, Berlin, Heidelberg (2009)
82. Watanabe, K., Fukushi, M., Horiguchi, S.: Collusion-resistant sabotage-tolerance mechanisms for volunteer computing systems. In: Proceedings of the 2009 IEEE International Conference on e-Business Engineering, ICEBE'09, pp. 213–218. IEEE Computer Society, Washington, DC, USA (2009). DOI 10.1109/ICEBE.2009.36
83. Wei, J., Zhang, X., Ammons, G., Bala, V., Ning, P.: Managing security of virtual machine images in a cloud environment. In: Proceedings of the ACM Cloud Computing Security Workshop, CCSW'09, pp. 91–96. ACM, New York, NY, USA (2009). DOI 10.1145/1655008.1655021
84. Wei, L., Zhu, H., Cao, Z., Jia, W., Vasilakos, A.: SecCloud: Bringing secure storage and computation in cloud. In: Proceedings of the IEEE International Conference on Distributed Computing Systems Workshops, ICDCSW'10, pp. 52–61. IEEE Computer Society, Washington, DC, USA (2010). DOI 10.1109/ICDCSW.2010.80
85. Wei, W., Du, J., Yu, T., Gu, X.: SecureMR: A service integrity assurance framework for MapReduce. In: Proceedings of the Annual Computer Security Applications Conference, ACSAC'09, pp. 73–82. IEEE Computer Society, Washington, DC, USA (2009). DOI 10.1109/ACSAC.2009.17
86. Xiao, Z., Xiao, Y.: P-accountable networked systems. In: Proceedings of the IEEE International Conference on Computer Communications Workshops (2010). DOI 10.1109/INFCOMW.2010.5466640
87. Xu, Y., Bailey, M., Jahanian, F., Joshi, K., Hiltunen, M., Schlichting, R.: An exploration of L2 cache covert channels in virtualized environments. In: Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW'11, pp. 29–40. ACM, New York, NY, USA (2011). DOI 10.1145/2046660.2046670
88. Yao, A.: How to generate and exchange secrets. In: Proceedings of the 27th Annual Symposium on Foundations of Computer Science, SFCS'86, pp. 162–167. IEEE Computer Society, Washington, DC, USA (1986). DOI 10.1109/SFCS.1986.25
89. Yumerefendi, A., Chase, J.: The role of accountability in dependable distributed systems. In: Proceedings of the 1st USENIX Conference on Hot Topics in System Dependability, HotDep'05, pp. 3–13. USENIX Association, Berkeley, CA, USA (2005)
90. Yumerefendi, A., Chase, J.: Strong accountability for network storage. Transactions on Storage **3**(3), 11:1–11:33 (2007). DOI 10.1145/1288783.1288786
91. Zalewski, M.: Browser security handbook. `http://bit.ly/gpBT9` (2009)
92. Zeng, K.: Publicly verifiable remote data integrity. In: Proceedings of the 10th International Conference on Information and Communications Security, ICICS'08, pp. 419–434. Springer-Verlag, Berlin, Heidelberg (2008). DOI 10.1007/978-3-540-88625-9_28
93. Zhang, K., Zhou, X., Chen, Y., Wang, X., Ruan, Y.: Sedic: Privacy-aware data intensive computing on hybrid clouds. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS'11, pp. 515–526. ACM, New York, NY, USA (2011). DOI 10.1145/2046707.2046767
94. Zheng, Q., Xu, S.: Fair and dynamic proofs of retrievability. In: Proceedings of the 1st ACM Conference on Data and Application Security and Privacy, CODASPY'11, pp. 237–248. ACM, New York, NY, USA (2011). DOI 10.1145/1943513.1943546
95. Zunnurhain, K., Vrbsky, S.: Security attacks and solutions in clouds. Poster from CloudCom 2010 (2010). URL `http://bit.ly/q1B9tJ`