Securing Resources in Decentralized Cloud Storage

Article *in* IEEE Transactions on Information Forensics and Security · May 2019

DOI: 10.1109/TIFS.2019.2916673

6 authors, including:

Enrico Bacis
University of Bergamo
21 PUBLICATIONS **202** CITATIONS
SEE PROFILE

Marco Rosa
SAP Research
20 PUBLICATIONS **125** CITATIONS
SEE PROFILE

Securing Resources in Decentralized Cloud Storage

Enrico Bacis*, Sabrina De Capitani di Vimercati†, Sara Foresti†,
Stefano Paraboschi*, Marco Rosa*, Pierangela Samarati†
*Università degli Studi di Bergamo, 24044 Dalmine - Italy, Email: *firstname.lastname*@unibg.it
†Università degli Studi di Milano, 20133 Milan - Italy, Email: *firstname.lastname*@unimi.it

***Abstract*—Decentralized Cloud Storage services represent a promising opportunity for a different cloud market, meeting the supply and demand for IT resources of an extensive community of users. The dynamic and independent nature of the resulting infrastructure introduces security concerns that can represent a slowing factor towards the realization of such an opportunity, otherwise clearly appealing and promising for the expected economic benefits. In this paper, we present an approach enabling resource owners to effectively protect and securely delete their resources while relying on decentralized cloud services for their storage. Our solution combines All-Or-Nothing-Transform for strong resource protection, and carefully designed strategies for slicing resources and for their decentralized allocation in the storage network. We address both availability and security guarantees, jointly considering them in our model and enabling resource owners to control their setting.**

***Index Terms*—Decentralized Cloud Storage; Secure deletion; Slicing and allocation; Security; Availability; Replication.**

I. Introduction

A clear recent trend in information technology is the rent by many users and enterprises of the storage/computation services from other parties. With cloud technology, what was in the past managed autonomously now sees the involvement of servers, often in an unknown location, immediately reachable wherever an Internet connection is present. Today the use of these Internet services typically assumes the presence of a Cloud Service Provider (CSP) managing the service. There are a number of factors that explain the current status. In general, the procurement and management of IT resources exhibit significant scale economies, and large-scale CSPs can provide services at costs that are less than those incurred by smaller players. Still, many users have an excess of computational, storage, and network capacity in the systems they own and they would be interested in offering these resources to other users in exchange of a rent payment. In the classical behavior of markets, the existence of an infrastructure that supports the meeting of supply and demand for IT services would lead to a significant opportunity for the creation of economic value from the use of otherwise under-utilized resources.

This change of landscape is witnessed by the increasing attention of the research and development community toward the realization of *Decentralized Cloud Storage* (DCS) services, characterized by the availability of multiple nodes that can be used to store resources in a decentralized manner. In such services, individual resources are fragmented in *shards* allocated (with replication to provide availability guarantees) to different nodes. Access to a resource requires retrieving all its shards. The main characteristics of a DCS is the cooperative and dynamic structure formed by independent nodes (providing a multi-authority storage network) that can join the service and offer storage space, typically in exchange of some reward. This evolution has been facilitated by blockchain-based technologies providing an effective low-friction electronic payment system supporting the remuneration for the use of the service. On platforms such as *Storj* [1], *SAFE Network Vault* [2], [3], *IPFS* [4], and *Sia* [5], users can rent out their unused storage and bandwidth to offer a service to other users of the network, who pay for this service with a network crypto-currency [6].

However, if security concerns and perception of (or actual) *loss of control* have been an issue and slowing factor for centralized clouds, they are even more so for a decentralized cloud storage, where the dynamic and independent nature of the network may hint to a further decrease of control of the owners on where and how their resources are managed. Indeed, in centralized cloud systems, the CSP is generally assumed to be honest-but-curious and is then trusted to perform all the operations requested by authorized users (e.g., delete a file when requested by the owner) [7]. The CSP is discouraged to behave maliciously, since this would clearly impact its reputation. On the contrary, the nodes of a decentralized system may behave maliciously when their misbehavior can provide economic benefits without impacting reputation (e.g., sell the content of deleted files). Client-side encryption typically assumed in DCSs provides a first crucial layer of protection, but it leaves resources exposed to threats, especially in the long term. For instance, resources are still vulnerable in case the encryption key is exposed, or in case of malicious nodes not deleting their shards upon the owner's request to try reconstructing the resource in its entirety.

Protection of the encryption key is therefore not sufficient in DCS scenarios, as it remains exposed to the threats above. A general security principle is to rely on more than one layer of defense. In this paper, we propose an additional and orthogonal layer of protection, which is able to mitigate these risks.

On the positive side, however, we note that the decentralized nature of DCS systems also increases the reliability of the service, as the involvement of a collection of independent parties reduces the risk that a single malfunction can limit the accessibility to the stored resources. In addition to this, the independent structure characterizing DCS systems - if coupled with effective resource protection and careful allocation to nodes in the network - makes them promising for actually strengthening security guarantees for owners relying on the decentralized network for storing their data.

In this paper, we present a solution to enable resource

owners to securely store their resources in DCS services, to share them with other users, while still being able to securely delete them. Our contribution is threefold. First, leveraging the protection guarantees offered by *All-Or-Nothing-Transform* (AONT), we devise an approach to carefully control *resource slicing* and *allocation* to nodes in the network, with the goal of ensuring both *availability* (i.e., retrieval of all slices to reconstruct the resource) and *security* (i.e., protection against malicious parties jointly collecting all the slices composing a resource). The proposed solution also enables the resource owners to securely delete their resources when needed, even when some of the nodes in the DCS misbehave. Second, we investigate different strategies for slicing and distributing resources across the decentralized network, and analyze their characteristics in terms of availability and security guarantees. Third, we provide a modeling of the problem enabling owners to control the granularity of slicing and the diversification of allocation to ensure the aimed availability and security guarantees. We demonstrate the effectiveness of the proposed model by conducting several experiments on an implementation based on an available DCS system. Our solution provides an effective approach for protecting data in decentralized cloud storage and ensures both availability and protection responding to currently open problems of emerging DCS scenarios, including secure deletion. In fact, common secret sharing solutions (e.g., Shamir [8]), while considering apparently similar requirements are not applicable in scenarios where the whole resource content (and not simply the encryption key) needs protection, because of their storage and network costs (e.g., each share in Shamir's method has the same size as the whole data that has to be protected).

Outline. The remainder of the paper is organized as follows. Section II introduces the basic concepts. Section III defines the properties of a decentralized allocation function with respect to replication and protection. Section IV discusses slicing and allocation strategies. Section V illustrates availability and security guarantees and discusses the setting of parameters guiding slicing and allocation. Section VI illustrates the implementation of our approach on a real DCS service and presents experimental results. Section VII discusses related work. Finally, Section VIII concludes the paper. The proofs of theorems are provided in Appendix A.

II. Basic Concepts and Scenario

The basic building block enabling the development of our solution is the application, at the client-side, of an *All-Or-Nothing-Transform* (AONT) encryption mode that transforms resources for their external storage. This mode requires the use of an encryption key. The encryption driven by the key represents the primary protection, and the use of AONT encryption mode further strengthens security. An AONT-encryption mode transforms a plaintext resource (original content in whatever form) into a ciphertext, with the property that the whole result of the transformation is required to obtain back the original plaintext. AONT guarantees in fact complete interdependence (*mixing*) among the bits of the encrypted resource in such a way that the unavailability of a portion of the encrypted resource prevents the reconstruction of any portion of the original plaintext. A party having access to a portion of the encrypted resource (but not to the encrypted resource in its entirety): *i)* if knowing the encryption key, it will not be able to reconstruct any portion of the resource (i.e., it will not be able to derive any information from the AONT-encrypted portions it has; the only option would be to attempt a brute force attack on the possible configurations of the missing portions, but their possible large size makes this attack unfeasible); *ii)* if not knowing the encryption key, it will not be able to perform brute-force attacks for guessing such a key, as any key (even the correct one) will be ineffective if not applied to the complete resource. AONT protection schemes can be built with the use of common cryptographic functions, like symmetric encryption and hash functions. An example of AONT scheme that guarantees complete mixing, which has also been used in the implementation of our prototype, is Mix&Slice [9]. Intuitively, Mix&Slice works by applying different rounds of encryption, each operating on a carefully designed combination of the bits resulting from the previous round. With Mix&Slice, i rounds of encryption working on blocks including b mini-blocks each, guarantee complete mixing of a resource composed of b^i mini-blocks. Figure 1 illustrates an example of mixing with two rounds of encryption. The first round mixes contiguous mini-blocks, while the second round mixes mini-blocks representatives of the different computations in the first round, providing a mixing of the whole resource content (as visible from the pattern-coding in the figure). Mix&Slice guarantees that each bit in the encrypted resource depends on the value of each bit in its plaintext representation. In our context, the use of AONT guarantees protection to the individual slices (and shards) composing the resource, and therefore to the resource itself (in its entirety as well as any of its portions). In fact, AONT makes each portion of the resource needed, in terms of information theory, to reconstruct any of the portions of the resource. The protection is then provided by the absence of information content.

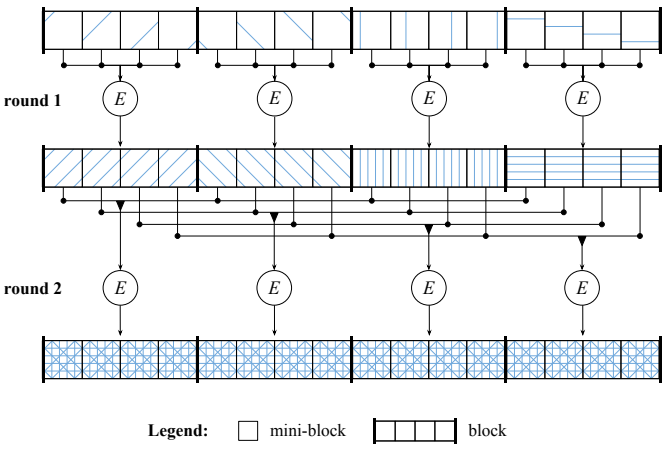


Fig. 1. An example of application of Mix&Slice

Figure 2 illustrates our reference scenario. The focus of this paper is the design of proper *slicing* of resources and

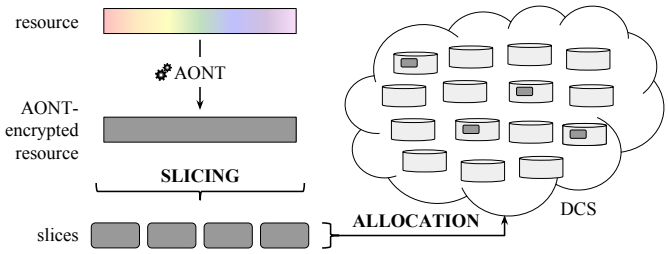


Fig. 2. Reference scenario

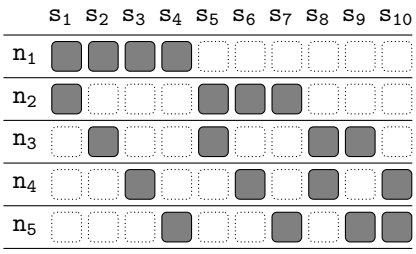


Fig. 3. An example of a minimal 3-protected and 2-replicated allocation function

the *allocation* of the produced slices to different nodes in the DCS system. Note that in the paper we use the term *slicing* to refer to the cutting of a resource and the term *slices* to refer to the result of such a process. A slice is therefore a chunk of the resource and represents a unit of allocation, in contrast to a shard that represents a portion of the resource allocated to a node (a shard can include several slices). Our approach focuses on *slicing* and *allocation* and is agnostic with respect to the specific AONT technique to be used, as long as the aimed strong protection guarantees are ensured, and with respect to the specific DCS adopted.

III. Allocation Properties

In our approach, the slicing of the resources into several slices to be distributed at the different nodes is guided by the availability and protection properties that need to be guaranteed. Availability (despite nodes failure or temporary unreachability) is provided through replication, security is provided through protection against malicious coalitions. Malicious nodes (and coalitions thereof) are interested in making the resource unavailable, by not returning the slices of the resource they store, or in providing access to a resource even after its deletion, by not removing the slices of the resource they store and returning such slices to (not authorized) users who pay for it. Before addressing slicing, we then characterize the replication and coalition resistance properties of the distribution of a resource.

We assume a (transformed) resource that has undergone AONT encryption (as described in the previous section) at the client side. For simplicity, we will omit such an explicit remark on transformation and we will simply use the term resource to denote an AONT-encrypted resource. Also, we assume a resource to be composed of different slices, for distribution in a DCS. We will address the problem of producing such slices in Section IV.

We model a resource as a set $\mathcal{S} = \{\mathtt{s}_1, \ldots, \mathtt{s}_\mathtt{s}\}$ of slices to be allocated to the nodes, denoted $\mathcal{N}$, of the DCS. The following definition formalizes slice allocation.

Definition 1 (Allocation function): Let $\mathcal{S}$ be a set of slices composing a resource and $\mathcal{N}$ be a set of nodes. An *allocation function* $\varphi : \mathcal{S} \rightarrow 2^{\mathcal{N}} \setminus \emptyset$ assigns each slice $\mathtt{s}_\mathtt{i} \in \mathcal{S}$ to a set of nodes $\varphi(\mathtt{s}_\mathtt{i}) = \mathtt{N}_\mathtt{i} \subseteq \mathcal{N}$, $\mathtt{N}_\mathtt{i} \neq \emptyset$.

The allocation function dictates how slices are allocated to nodes in the DCS. The consideration of sets of nodes (in contrast to individual nodes) in the co-domain accommodates replication. The exclusion of the empty set of nodes ensures lossless distribution (i.e., each slice is allocated to at least one node). Figure 3 illustrates an example of an allocation function, considering a resource split into ten slices ($\mathcal{S} = \{\mathtt{s}_1, \ldots, \mathtt{s}_{10}\}$) allocated to five nodes ($\mathtt{n}_1, \ldots, \mathtt{n}_5$) in the DCS (nodes not used in the allocation are not reported in the figure). The figure has a row for each node and a column for each slice. The allocation of a slice to a node is represented by a gray box at the intersection between the row representing the node and the column representing the slice. Empty boxes with a dotted frame represent the fact that the slice is not allocated to the node. For example, $\varphi(\mathtt{s}_1) = \{\mathtt{n}_1, \mathtt{n}_2\}$.

We identify two main properties of an allocation, characterizing the availability, provided by *replication*, and the *protection* against possible malicious coalitions of nodes, provided by the diversification of the allocation.

We characterize availability provided by replication in terms of the number of replicas maintained in the system. While in principle the number of replicas maintained for each slice can differ, we assume the same number of replicas is used for all the slices. This derives from the fact that we assume that nodes are not associated with individual reliability profiles (Section V). Since all slices are needed to reconstruct the resource, using fewer replicas for any of the slices would decrease the availability of the resource, which will be dictated by such a lower bound. The following definition formalizes the replication degree of an allocation function.

Definition 2 (r-Replicated allocation function): Let $\mathcal{S}$ be a set of slices composing a resource, $\mathcal{N}$ be a set of nodes, and φ be an allocation function. Function φ is *r-replicated* iff $\forall \mathtt{s}_\mathtt{i} \in \mathcal{S}$, $|\varphi(\mathtt{s}_\mathtt{i})| \geq r$.

For instance, the allocation function in Figure 3 is 2-replicated, as two copies are maintained for each slice.

We characterize the protection offered by an allocation in terms of the minimum number of nodes required to reconstruct a resource, as formalized by the following definition.

Definition 3 (k-Protected allocation function): Let $\mathcal{S}$ be a set of slices composing a resource, $\mathcal{N}$ be a set of nodes, and φ be an allocation function. Function φ is *k-protected* iff for each $\mathtt{N}_\mathtt{i} \subset \mathcal{N}$, with $|\mathtt{N}_\mathtt{i}| \leq k$, $\exists \mathtt{s}_\mathtt{j} \in \mathcal{S}$ s.t. $\varphi(\mathtt{s}_\mathtt{j}) \cap \mathtt{N}_\mathtt{i} = \emptyset$.

A k-protected allocation function guarantees distribution of slices to nodes in such a way to dictate the cooperation of no less than $k+1$ nodes to collect all the slices composing the resource (and hence enabling retrieving its plaintext). In other

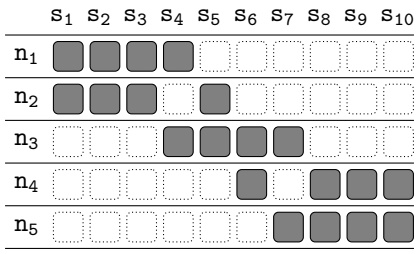


Fig. 4. An example of 2-replicated allocation function that is not 3-protected

words, a k-protected allocation function guarantees protection of the resource against malicious (i.e., colluding) behavior of up to k nodes. In fact, with a k-protected allocation function, for each coalition of k nodes in $\mathcal{N}$, there is at least a slice that is not stored at any of the nodes in the coalition. Hence, such a coalition can neither decrypt the resource with a brute-force attack, nor prevent its deletion. The allocation function in Figure 3 is 3-protected: any subset of 3 out of the 5 nodes misses at least a slice. For instance, coalition $\{\mathtt{n}_1,\mathtt{n}_2,\mathtt{n}_3\}$ misses slice $\mathtt{s}_{10}$, while coalition $\{\mathtt{n}_1,\mathtt{n}_2,\mathtt{n}_4\}$ misses slice $\mathtt{s}_9$. On the contrary, the allocation function in Figure 4, on the same slices and nodes, is not 3-protected (but only 2-protected): coalition $\{\mathtt{n}_1,\mathtt{n}_3,\mathtt{n}_4\}$ jointly possesses all the slices.

We refer to an allocation function that is r-replicated, according to Definition 2, and k-protected, according to Definition 3, as a (k,r)-*allocation*.

Definition 4 ((k,r)-Allocation): Let $\mathcal{S}$ be a set of slices composing a resource, $\mathcal{N}$ be a set of nodes, and φ be an allocation function. Function φ is a (k,r)-*allocation* iff it is k-*protected* and r-*replicated*.

According to Definitions 2 and 3, a (k,r)-allocation is also a (k',r')-allocation, for any $r' \leq r$ and any $k' \leq k$. In fact, trivially, an allocation function providing r replicas also provides $r' < r$ replicas. Analogously, an allocation function protecting a resource from coalitions of k nodes also protects the resource from coalitions of $k' < k$ nodes. Among all (k,r)-allocations, we are interested in identifying those for which k and r represent the highest values satisfying the availability and protection properties (i.e., satisfying the properties in a minimal way). We call such allocation functions minimal, as formalized by the following definition.

Definition 5 (Minimal (k,r)-allocation): Let $\mathcal{S}$ be a set of slices composing a resource, $\mathcal{N}$ be a set of nodes, and φ be a (k,r)-allocation. Function φ is *minimal* iff:

1) it is not $(k+1)$-protected;
2) $\forall \mathtt{s_i} \in \mathcal{S}$, $|\varphi(\mathtt{s_i})| = r$.

According to Definition 5, a *minimal* (k,r)-allocation is an allocation that guarantees protection against coalitions of up to k (but no more) nodes and that uses exactly r replicas. The allocation function in Figure 3 is an example of minimal $(3,2)$-allocation. In the following, we will restrict our attention to minimal allocation functions and, when talking about a (k,r)-allocation, we will implicitly assume such minimality.

IV. Slicing and Allocation Strategies

In the absence of replication, producing an allocation that guarantees k-protection, that is, a $(k,1)$-allocation, is straightforward: it is sufficient to split the resource into $k+1$ slices and allocate each slice to a different node. When considering replication, different approaches can be taken for allocation, differing in the granularity of slicing and in how allocation diversifies the storage at different nodes. In the following, we discuss these options. In the discussion, in addition to parameters k and r introduced before, we will use parameters s, denoting the number of slices in which a resource is split, and n, denoting the number of nodes to be involved in the allocation of a resource. Different approaches vary in the number s of slices to be considered and in the number n of nodes to be involved for providing a (k,r)-allocation. We note that, with respect to nodes, the only parameter to be considered in the allocation strategies is the number n of nodes to be involved (the specific nodes to be involved can be selected randomly). We identify and study the behavior of two approaches for producing a (k,r)-allocation. The first approach aims to minimize the number of slices (*Min_slices*), while the second aims to minimize the number of nodes (*Min_nodes*). We analyze these two approaches as they represent the two extremes with respect to granularity of slicing and diversification of allocation. Their analysis permits to highlight the characteristics of fine-grained (*Min_nodes*) and coarse-grained (*Min_slices*) slicing, and can also represent a reference for intermediate configurations.

A. Minimizing the number of slices

We start noting that the number s of slices involved for guaranteeing a (k,r)-allocation must be such that $s \geq k+1$. In fact, there should be at least $k+1$ slices to guarantee k-protection, as formally captured by the following theorem.

Theorem 1 (Minimum number of slices): Let k be a protection parameter and r be a replication factor. The number s of slices necessary to define a (k,r)-allocation is $s \geq k+1$.

A simple approach for determining a (k,r)-allocation extends the natural approach of producing $k+1$ slices, by simply considering their replication at different nodes. Such an approach is characterized by a *coarse-slicing*, since minimizing the number of slices clearly entails a larger size for them, and by *consistent replication* (i.e., nodes have no intersection or complete intersection of stored slices).

We observe that a (k,r)-allocation function using the minimum number ($s = k+1$) of slices implies that:

1) a node maintains at most one slice, that is, $|\varphi^{-1}(\mathtt{n_i})| = 1$, $\forall \mathtt{n_i} \in \mathcal{N}$ involved in the allocation;
2) the number of nodes involved in the allocation is exactly r times the number of slices, that is, $n = r \cdot (k+1)$.

The first observation derives from the fact that, since there are only $k+1$ slices, placing more than one slice on a node would imply the existence of a set of k nodes able to reconstruct the resource and therefore would not guarantee k-protection anymore. The second observation naturally derives

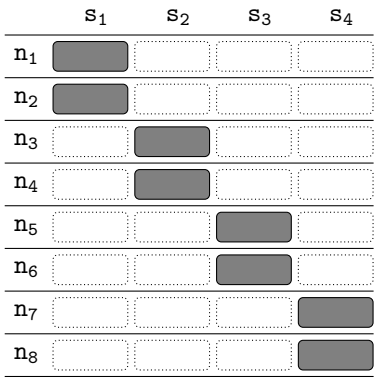


Fig. 5. An examples of $(3, 2)$-allocation that minimizes the number of slices

from the first, considering that every slice needs to be replicated r times. The following theorem proves the observations above.

Theorem 2: Let k be a protection parameter and r be a replication factor. A (k, r)-allocation $\varphi : \mathcal{S} \rightarrow 2^{\mathcal{N}} \setminus \emptyset$ that adopts the minimum number of slices $s = k + 1$ is such that:

1) $|\varphi^{-1}(\mathtt{n_i})| = 1$, $\forall \mathtt{n_i} \in \mathcal{N}$ involved in the allocation;
2) the number of nodes involved in the allocation is $n = r \cdot (k + 1)$.

As an example, a $(3, 2)$-allocation using the minimum number of slices would imply splitting the resources into 4 $(= 3 + 1)$ slices, generating 2 copies of each slice, to be distributed at 8 different nodes. Figure 5 illustrates an example of allocation function enforcing this.

A (k, r)-allocation that uses the minimum number of slices $s = k + 1$ well resists to failures. Indeed, $k + 1$ nodes out of $r \cdot (k + 1)$ are sufficient to reconstruct the resource content, as long as one replica of each slice is available. However, the number of nodes used by such an allocation function quickly grows with k and r. For instance, a $(10, 5)$-allocation would need 55 $(= 5 \cdot (10 + 1))$ nodes.

B. Minimizing the number of nodes

At the other end of the spectrum of possible strategies for defining and distributing slices to guarantee a (k, r)-allocation, there are functions minimizing the number of nodes to be involved in the distribution (and deriving the number of slices in which the resource needs to be split based on this).

A trivial lower bound on the number of nodes that need to be involved in a (k, r)-allocation is $n \geq max(k + 1, r)$, since there should be at least r nodes to hold r replicas and at least $k + 1$ nodes to guarantee k-protection. The minimum number of nodes to be involved to guarantee (k, r)-allocation is actually higher than that as it needs to be at least the sum of the protection and replication parameters (k and r), as stated by the following theorem.

Theorem 3 (Minimum number of nodes): Let k be a protection parameter and r be a replication factor. The number n of nodes necessary to define a (k, r)-allocation is $n \geq k + r$.

The minimum number of nodes stated by Theorem 3 derives from two simple observations. First, to guarantee k-protection, for each coalition of k nodes, there must exist at least one slice that is not stored at any of the nodes in the coalition. Second, to provide r-replication, such a slice should be stored at (at least) r nodes that are not in the coalition. Hence, at least $k + r$ nodes need to be involved. As we will illustrate in the following, $k + r$ nodes, besides been necessary, are also sufficient to define a (k, r)-allocation.

While using the minimum number of slices applies a coarse slicing with consistent replication, using the minimum number of nodes applies a *fine-grained slicing* with *diversified replication* across nodes. Intuitively, instead of splitting the resource into slices and allocating to each node a single slice, minimizing the number of nodes requires slicing the resource into more fine-grained slices and allocating the slices to nodes in a diversified manner, to guarantee that no set of k nodes jointly possesses all the slices. The definition of the allocation requires then to identify the number of slices in which a resource needs to be split, which must be sufficient to distribute the r replicas to nodes while ensuring k-protection. The minimum number of slices needed for ensuring that no set of k nodes is able to reconstruct the resource when using $k + r$ nodes, clearly happens when any set of k nodes misses exactly one slice (which, given r-replication, would instead be stored at the r nodes not belonging to the set) and no two coalitions miss the same slice. In fact, if two sets of k nodes miss the same slice, such a slice could not have r replicas when using only $k + r$ nodes. The number of required slices can then be identified as the number of coalitions of k nodes out of $k+r$, that is $\binom{k+r}{k}$, as formally proved by the following theorem.

Theorem 4: Let k be a protection parameter and r be a replication factor. Each (k, r)-allocation that adopts the minimum number of nodes $n = k + r$ uses $s = \binom{n}{k} = \binom{k+r}{k}$ slices.

A (k, r)-allocation that uses $k+r$ nodes and $\binom{k+r}{k}$ slices has two interesting properties. The first one, already noted, is that any coalition of k nodes *misses exactly one* slice. The second one, deriving from the fact that the missing slice is different for different coalitions, is that *any* set of $k + 1$ nodes is sufficient to reconstruct the resource (differently from the *Min_slices* approach where at least $k + 1$ nodes are needed to reconstruct the resource but not any set of $k + 1$ nodes guarantees that). The following theorem proves these two properties.

Theorem 5: Let k be a protection parameter and r be a replication factor. Each (k, r)-allocation that adopts the minimum number of nodes $n = k + r$ and $s = \binom{k+r}{k}$ slices guarantees that:

1) $\forall \mathtt{N_i} \subset \mathcal{N}$ with $|\mathtt{N_i}| = k$, $\exists! \mathtt{s_j}$ s.t. $\varphi(\mathtt{s_j}) \cap \mathtt{N_i} = \emptyset$;
2) $\forall \mathtt{N_i} \subseteq \mathcal{N}$ with $|\mathtt{N_i}| = k + 1$, $\bigcup_{\mathtt{n_j} \in \mathtt{N_i}} \varphi^{-1}(\mathtt{n_j}) = \mathcal{S}$.

A (k, r)-allocation that minimizes the number of nodes can be obtained by assuming $\mathcal{N}$ to comprise $k + r$ nodes and proceeding as follows. Let $2^{\mathcal{N}}_k = \{\mathtt{N_i} \in 2^{\mathcal{N}} : |\mathtt{N_i}| = k\}$ be all subsets of k nodes in $\mathcal{N}$. For each slice $\mathtt{s_i} \in \mathcal{S}, i = 1, \ldots, \binom{k+r}{k}$, $\varphi(\mathtt{s_i}) = \{\mathcal{N} \setminus \{\mathtt{N_i}\} : \text{with } \mathtt{N_i} \in 2^{\mathcal{N}}_k\}$. Intuitively, for each slice $\mathtt{s_i}$, $\varphi(\mathtt{s_i})$ selects a coalition of k nodes that misses $\mathtt{s_i}$ and allocates slice $\mathtt{s_i}$ to all the other nodes. This guarantees that each coalition $(\mathtt{N_i})$ of k nodes misses at

least one slice ($\mathtt{s_i}$), providing k-protection. Slice $\mathtt{s_i}$, which represents the missing slice for coalition $\mathtt{N_i}$, is stored at all the other $n-k=r$ nodes in $\mathcal{N}$, providing r-replication. Intuitively, in a (k,r)-allocation using the minimum number of nodes, no two slices are allocated exactly to the same set of nodes (i.e., $\forall \mathtt{s_i}, \mathtt{s_j} \in \mathcal{S}$, $\varphi(\mathtt{s_i}) \neq \varphi(\mathtt{s_j})$). In fact, the possible subsets of r nodes in $\mathcal{N}$ is $\binom{k+r}{r}$ and $\binom{k+r}{k} = \binom{k+r}{r}$.

For example, a $(3,2)$-allocation using the minimum number of nodes requires $n = k+r = 3+2 = 5$ nodes and the use of $s = \binom{k+r}{k} = \binom{5}{3} = 10$ slices. Figure 3 illustrates an example of $(3,2)$-allocation distributing 10 slices over 5 nodes. The allocation is a $(3,2)$-allocation since it replicates each slice twice while guaranteeing that no coalition of 3 nodes possesses all the slices. More precisely, any coalition of 3 nodes misses exactly one slice and the missing slice is different for any of such coalitions. For instance, coalition $\{\mathtt{n_1}, \mathtt{n_2}, \mathtt{n_3}\}$ misses slice $\mathtt{s_{10}}$, while coalition $\{\mathtt{n_1}, \mathtt{n_2}, \mathtt{n_4}\}$ misses slice $\mathtt{s_9}$.

C. Discussion

We have discussed two alternative strategies for producing a (k,r)-allocation, aimed to minimize the number of slices (*Min_slices*, with coarse-grained slicing and consistent replication) and to minimize the number of involved nodes (*Min_nodes*, with fine-grained slicing and diversified replication). When no replication is used (i.e., $r=1$) these two strategies are equivalent, as each would imply the use of the same number of slices $s = k+1$ and nodes $n = k+1$. On the contrary, when replication is adopted (i.e., $r > 1$) the two strategies differ in the number of nodes n and slices s used and in the distribution of slices to nodes. Besides these two extreme configurations, the resource owner can decide to adopt other allocation strategies. The analysis presented in this section can then represent a reference for the definition and analysis of intermediate configurations.

We note that the structure of the *Min_nodes* strategy has a correspondence with *secret sharing* [8]. In (m,d) secret sharing, the goal is to build d shares of a secret such that at least m of them are necessary to reconstruct a secret. Given a *Min_nodes* (k,r)-allocation, with $k+r$ shares, the use of a $(k+1,\ k+r)$ secret sharing scheme would then satisfy the requirement that at least $k+1$ nodes have to cooperate to access the resource, tolerating the loss of up to $r-1$ nodes. Compared to the well-known Shamir's technique for secret sharing [8], the approach we propose shows a significant advantage with respect to storage and network capacity. In terms of computational cost, Shamir's technique requires to identify the roots of a polynomial, while our approach requires the application of symmetric encryption algorithms. The performance of symmetric encryption algorithms is so high, particularly for algorithms implemented in hardware by the CPU, that the potentially simpler computational structure of Shamir's technique does not provide an advantage and turns out to be slower when considering large resources. However, the computational cost is in any case a marginal element in this domain. In terms of storage, Shamir's approach offers security if each of the shares has the same size as the secret. With a $(k+1,\ k+r)$ secret sharing scheme, there is therefore the need to store in the network $k+r$ times the amount of plaintext data, whereas our solution is characterized by the replication factor r. In terms of the minimum amount of data that has to be accessed by the owner, Shamir's solution asks the owner to read $k+1$ times the size of the plaintext, whereas our technique, if the storage nodes support access to portions of the resource, does not require to access more than the size of the plaintext. We can then conclude that Shamir's technique, which is quite interesting for domains where the secret has a small size (e.g., encryption keys), is not convenient in the domain considered in this work. When Shamir's method is used to protect only the encryption key and then encryption is used to protect the resource, Shamir's method can be assumed to be only a key management strategy, making encryption of the resource the only protection measure, without offering the level of protection provided by AONT.

Note also that, for simplicity, we have assumed that the owner can arbitrarily split her resource as needed for the definition of a (k,r)-allocation. However, thanks to its flexibility, our approach can be adopted also when the encrypted resource is already organized in *chunks* that cannot be split for allocation (e.g., blocks resulting from the AONT algorithm adopted), or in general when slicing is constrained. Indeed, even if in the discussion, for simplicity, we consider slices of equal size, our approach can be adopted also if the size varies. Also, slices can contain non-contiguous chunks of the resource. Clearly, the number of chunks should be sufficient for the definition of a (k,r)-allocation (e.g., $k+1$ and $\binom{n}{k}$ in our two alternative configurations). If the resource includes fewer chunks, it needs to be padded. If the resource includes more chunks than necessary, the resource owner can combine the chunks in s slices and apply the chosen allocation function over these slices. As an example, to define a (3,2)-allocation for a resource organized in 20 chunks using 5 nodes, chunks can be arbitrarily combined to identify 10 slices for allocation. Alternatively, k-protection and r-replication can be obtained by considering each chunk as a different slice and interpreting the allocation function as periodic in s, or simply by randomly allocating the chunks after the first s (which are the ones necessary to guarantee k-protection). For instance, a (3,2)-allocation for a resource with 20 chunks using 5 nodes can be obtained by applying the allocation function in Figure 3 twice (on slices $\mathtt{s_1}, \ldots, \mathtt{s_{10}}$ and $\mathtt{s_{11}}, \ldots, \mathtt{s_{20}}$), or by using it for slices $\mathtt{s_1}, \ldots, \mathtt{s_{10}}$ while arbitrarily allocating slices $\mathtt{s_{11}}, \ldots, \mathtt{s_{20}}$ at two nodes each.

V. Availability and Protection Guarantees

Parameters r and k introduced in the previous section characterize the degree of replication and of protection against malicious coalitions of nodes. Such parameters provide a clean and precise modeling and allow reasoning about properly setting the number of slices and the number of nodes to be involved in the allocation. The setting of k and r to provide given security and availability guarantees clearly depends on the specific characteristics of the network. For instance, in a stable network a low number of replicas may suffice to

provide high availability, while in a highly dynamic and non-resilient network a higher number of replicas should be used to enjoy the same guarantee. In the same vein, actual protection against possible exposure of a resource to malicious coalitions depends on the nature of nodes involved in the allocation. Consistently with these observations, we note that a natural way for the resource owner to express and reason about availability and protection guarantees is the probability of the resource to become unavailable and the probability of a coalition of malicious nodes to jointly possess all the resource slices. In this section, we illustrate how to derive proper r and k settings to be then used for splitting resources into slices and for slices allocation, starting from the aimed guarantee of availability and security expressed in terms of such probabilities. Clearly, the probability of a resource to become unavailable, or exposed to malicious coalitions, depends on the probability of individual nodes to become unavailable or behaving maliciously. We then introduce the probability of a single node to fail, and hence to become unavailable, denoted p_u, and the probability of a node to behave maliciously, and hence to participate in a malicious coalition compromising protection, denoted p_c. We assume, as common in decentralized systems, the probability p_u of failure to be the same for all nodes and the failure of any node to be not influenced by the failure of the other nodes. This assumption enables a clean modeling, which can be taken as a reference for reasoning on different probability distributions. Since the selection of storage nodes is driven by a pseudorandom function, we also consider a uniform probability p_c of compromise and assume independence of compromise events on different nodes. We introduce the probability of a resource to become unavailable, denoted P_u, and of being exposed to a malicious coalition, denoted P_c, when using a (k, r)-allocation. The analysis will then guide the identification of the values for k and r to be used to guarantee that P_u and P_c do not exceed a given threshold. We discuss separately the *Min_slices* and *Min_nodes* allocation strategies introduced in the previous section, which, as we will see, exhibit a different behavior with respect to availability and security guarantees.

A. Min_slices allocation

Using a (k, r)-allocation with the minimum number of slices, unavailability of a resource happens when, for any of the $k + 1$ slices composing the resources, all the r nodes storing the replica of the slice fail. The probability of such an event to happen is $P_u = 1 - (1 - (p_u)^r)^{k+1}$, where $(1 - (p_u)^r)$ is the probability that one of the r replicas of a slice is available and, for the assumption on the independence of the failure events, $(1-(p_u)^r)^{k+1}$ is the probability that one replica of each of the $k + 1$ slices is available. In the same vein, the resource becomes exposed (and hence a compromise happens and deletion cannot be guaranteed) when a coalition of malicious nodes collectively possesses all the $k + 1$ slices, that is, when the coalition contains $k+1$ nodes each possessing a different slice. The probability of such an event to happen is $P_c = (1-(1-p_c)^r)^{k+1}$, where $(1-p_c)^r$ is the probability that one replica is stored on a node that is not part of a coalition and, consequently, $1 - (1 - p_c)^r$ is the probability that one replica is exposed. Since such an exposure must involve all the $k + 1$ slices, the probability that a coalition possesses all the slices is $(1-(1-p_c)^r)^{k+1}$. The following theorem proves such observations.

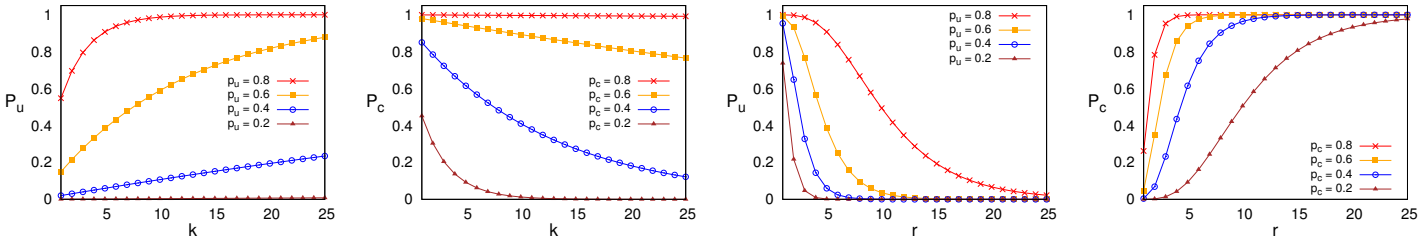
Theorem 6: Given a set $\mathcal{S}$ of slices composing a resource, a set $\mathcal{N}$ of nodes with probability of failure p_u and probability of being compromised p_c, and a (k, r)-allocation using the minimum number of slices:

- $P_u = 1 - (1-(p_u{}^r))^{k+1}$
- $P_c = (1 - (1 - p_c)^r)^{k+1}$

Figure 6 illustrates how k and r affect the values of P_u (Figure 6(a,c)) and P_c (Figure 6(b,d)), considering different values of p_u and p_c, respectively. The values considered for p_u and p_c are 0.2, 0.4, 0.6, and 0.8. These values, extremely pessimistic with respect to what can be expected in real systems, have been chosen to study the behavior of the probabilistic formulas. Figure 6(a) reports the values of P_u assuming a fixed number $r = 5$ of replicas and varying k between 1 and 25. Figure 6(c) reports the values of P_u assuming a fixed $k = 5$ and varying the number r of replicas between 1 and 25. Figures 6(b,d) report the values of P_c in the same settings of Figures 6(a,c). As it can be seen from Figure 6(a), P_u increases as the value of k increases, because the number of nodes used in the allocation increases and therefore the probability of availability of a larger number of slices decreases. Indeed, the number of nodes necessary to reconstruct a resource grows with k (it is $k + 1$), and the probability of availability of all the nodes necessary to reconstruct the resource decreases. However, P_u remains low if the failure probability of a single node p_u is low. Probability P_u instead decreases as the value of r increases (Figure 6(c)), because each slice will be stored on a larger number of nodes, reducing the risk of unavailability. Figure 6(b) shows that P_c decreases as k increases because the number of nodes that should be part of a coalition increases, meaning that the probability of forming a coalition decreases. Probability P_c increases as r increases (Figure 6(d)), because the number of replicas of each slice increases and therefore also the probability that one replica is stored on a compromised node increases.

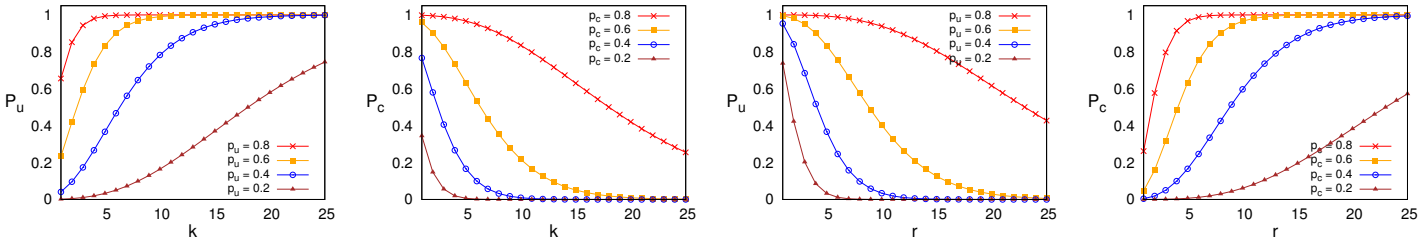
B. Min_nodes allocation

Using a (k, r)-allocation with the minimum number of nodes, the unavailability of the resource occurs when any combination of r (or more) nodes becomes unavailable. In fact, regardless of the slices that those nodes store, such an event causes at least one slice to be unavailable. The probability P_u that a resource becomes unavailable is then $P_u = \sum_{i=r}^{k+r} \binom{k+r}{i} (p_u)^i (1 - p_u)^{k+r-i}$, where the binomial coefficient $\binom{k+r}{i}$ is the number of all possible combinations of i nodes over $k+r$, with i varying in the range $r, \ldots, k+r$, that can be unavailable; $(p_u)^i$ is the probability that i nodes are unavailable; and $(1 - p_u)^{k+r-i}$ is the probability that the remaining nodes (i.e., $k+r-i$) are available. In the same vein, any coalition of $k+1$ nodes causes an exposure of the resource,



(a) $k = 1, \ldots, 25$, $r = 5$ (b) $k = 1, \ldots, 25$, $r = 5$ (c) $k = 5$, $r = 1, \ldots, 25$ (d) $k = 5$, $r = 1, \ldots, 25$

Fig. 6. Probability that the resource is unavailable (a,c) and that it is exposed (b,d) using a (k, r)-allocation that minimizes the number of slices, with r=5 varying k between 1 and 25 (a,b), and with k=5 varying r between 1 and 25 (c,d)



(a) $k = 1, \ldots, 25, r = 5$ (b) $k = 1, \ldots, 25, r = 5$ (c) $k = 5$, $r = 1, \ldots, 25$ (d) $k = 5$, $r = 1, \ldots, 25$

Fig. 7. Probability that the resource is unavailable (a,c) and that it is exposed (b,d) using a (k, r)-allocation that minimizes the number of nodes, with $r = 5$ varying k between 1 and 25 (a,b), and with $k = 5$ varying r between 1 and 25 (c,d)

regardless of the slices they store. Relying on the minimum number of nodes, in fact, implies that any coalition of $k + 1$ nodes possesses all the slices (Theorem 5). The probability P_c of a compromise is then $P_c = \sum_{i=k+1}^{k+r} \binom{k+r}{i} (p_c)^i (1-p_c)^{k+r-i}$, where the binomial coefficient $\binom{k+r}{i}$ is the number of all possible coalitions of i nodes over $k+r$ nodes, with i varying in the range $k + 1, \ldots, k + r$; $(p_c)^i$ is the probability that i nodes form a coalition; and $(1 - p_c)^{k+r-i}$ is the probability that the remaining nodes (i.e., $k+r-i$) are not compromised. The following theorem proves such observations.

Theorem 7: Given a set $\mathcal{S}$ of slices composing a resource, a set $\mathcal{N}$ of nodes with probability of failure p_u and probability of being compromised p_c, and a (k, r)-allocation using the minimum number of nodes:

- $P_u = \sum_{i=r}^{k+r} \binom{k+r}{i} (p_u)^i (1 - p_u)^{k+r-i}$
- $P_c = \sum_{i=k+1}^{k+r} \binom{k+r}{i} (p_c)^i (1 - p_c)^{k+r-i}$

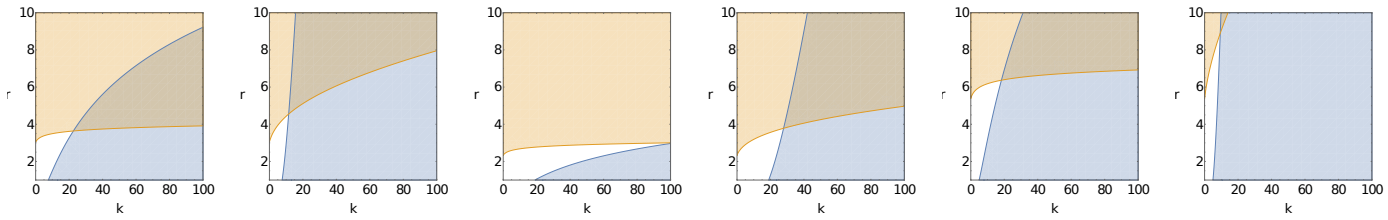
Figure 7 illustrates how k and r affect the values of P_u (Figures 7(a,c)) and P_c (Figures 7(b,d)), considering different values of p_u and p_c, respectively. The values considered for p_u and p_c are 0.2, 0.4, 0.6, and 0.8. Figure 7(a) reports the values of P_u assuming a fixed number $r = 5$ of replicas and varying k between 1 and 25. Figure 7(c) reports the values of P_u assuming a fixed $k = 5$ and varying the number r of replicas between 1 and 25. Figures 7(b,d) report the values of P_c in the same settings as Figures 7(a,c). From the figures, it is immediate to see that P_u and P_c present a similar behavior when adopting a configuration minimizing the number of slices and of nodes (i.e., P_u increases as k grows and decreases as r grows, while P_c decreases as k grows and increases as r grows).

C. Setting k and r

Our modeling of the probability that a resource is not available (P_u) and that it is exposed (P_c) can be used to set appropriate values for parameters k and r. To this purpose, fixing the maximum threshold P_u^{max} of resource unavailability and P_c^{max} of resource exposure, we compute all the configurations of k and r that guarantee $P_u \leq P_u^{max}$ and $P_c \leq P_c^{max}$ through the formulas in Theorems 6 and 7. Clearly, the values of k and r for the configurations satisfying the thresholds depend on the chosen allocation function.

Comparing the evolution of the probability P_u that a resource becomes unavailable using the *Min_slices* and *Min_nodes* allocation strategies, varying k (Figure 6(a) and Figure 7(a)), we can easily see that *Min_slices* is more robust against node failure (i.e., P_u increases slowly) than *Min_nodes*. This is due to the fact that even if the number of nodes involved in the allocation increases in both configurations, with an allocation that minimizes the number of nodes the impact of a node failure on the availability of the resource is significant. A similar comment applies when comparing how P_u evolves in the two configurations varying the number r of replicas (Figure 6(c) and Figure 7(c)). In this case, the decrease of P_u with *Min_slices* is faster than the decrease of P_u with *Min_nodes*. Therefore, we can conclude that, for configurations with the same values for r and k, *Min_slices* exhibits higher availability.

Comparing the evolution of the probability P_c that a resource is exposed due to a coalition of at least $k + 1$ nodes using *Min_slices* and *Min_nodes* allocation strategies, varying k (Figure 6(b) and Figure 7(b)), we can easily see that *Min_nodes* is more robust (i.e., P_c decreases faster) than *Min_slices*. This is due to the fact that, with an allocation that minimizes the number of nodes, the probability of forming a coalition of at least $k + 1$ nodes among the $k + r$ nodes



(a) *Min_slices*, p_u=0.005, p_c=0.2 (b) *Min_nodes*, p_u=0.005, p_c=0.2 (c) *Min_slices*, p_u=0.001, p_c=0.5 (d) *Min_nodes*, p_u=0.001, p_c=0.5 (e) *Min_slices*, p_u=0.05, p_c=0.1 (f) *Min_nodes*, p_u=0.05, p_c=0.1

Fig. 8. *Min_slices* and *Min_nodes* (k, r)-allocations that guarantee $P_u \leq 10^{-7}$ and $P_c \leq 10^{-6}$ with different values for p_u and p_c

is smaller than the probability of controlling at least one of the r nodes for each of the $k+1$ slices of the allocation that minimizes the slices. A similar comment applies when comparing how P_c evolves in the two configurations varying the number of replicas (Figure 6(d) and Figure 7(d)). The increase of probability P_c using *Min_slices* is faster than the increase of P_c using *Min_nodes*, because it is more difficult to control at least $k+1$ of the $k+r$ nodes than to control at least one node in each of the distinct $k+1$ groups of r nodes. Therefore, we can conclude that, for configurations with the same values for r and k, *Min_nodes* exhibits higher security.

When the resource owner has chosen the preferred allocation function, given the maximum threshold P_u^{max} of resource unavailability and P_c^{max} of resource exposure, different configurations of k and r guarantee that $P_u \leq P_u^{max}$ and $P_c \leq P_c^{max}$. Among all these configurations, the ones with low replication factor (r) require less storage and have lower economic costs, while the ones involving a limited number (n) of nodes enjoy simplicity in the management of the system and better performance of access operations (less connections have to be established). Figure 8 considers three different network configurations, characterized by a different probability p_u for single nodes to fail and a different probability p_c to behave maliciously, and illustrates the configurations of k and r satisfying the above thresholds using *Min_slices* and *Min_nodes* allocation strategies. In the figure, the orange area on the top-left represents the configurations of k and r that satisfy the availability requirement (i.e., $P_u \leq 10^{-7}$), while the blue area on the bottom-right represents the configurations that satisfy the security requirement (i.e., $P_c \leq 10^{-6}$). We chose these thresholds because the overall availability guarantee ($P_u^{max} = 10^{-7}$) is the same declared in the specification of the system used in our experiments (i.e., Storj). We chose a higher value for P_c^{max} than P_u^{max} because protection against coalitions represents a security layer adding to the protection already offered by encryption. The intersection between the orange and blue areas represents configurations that provide both availability and security guarantees within the thresholds set by the owner. Among these configurations, the one located on the left/bottom corner of the intersecting area is the one to be preferred as the number of nodes and replicas is minimum.

Figures 8(a,b) consider nodes with $p_u = 0.005$ and $p_c = 0.2$. The optimal configuration for *Min_slices* it is $k = 26$ and $r = 4$ (i.e., $n = 108$), while for *Min_nodes* allocation is $k = 12$ and $r = 5$ (i.e., $n = 17$). The second allocation, although more expensive on storage, due to one additional replica, considerably reduces the number of nodes involved in the storage of the resource compared to the adoption of the first allocation function. Our analysis demonstrates that this is a general behavior: *Min_nodes* requires the same (or a slightly higher) number r of replicas and a significantly lower number n of nodes than *Min_slices*. This observation is confirmed by the extreme scenarios illustrated in Figures 8(c,d), considering highly reliable ($p_u = 0.001$) but lowly trusted ($p_c = 0.5$) nodes, and in Figures 8(e,f), considering unreliable ($p_u = 0.05$) but relatively trusted ($p_c = 0.1$) nodes. The optimal configurations in Figures 8(c,d) are $k = 100$ and $r = 3$ for *Min_slices* (i.e., $n = 303$), and $k = 27$ and $r = 4$ for *Min_nodes* (i.e., $n = 31$, meaning that the number of nodes is ten times smaller). The optimal configurations in Figures 8(e,f) are $k = 10$ and $r = 9$ (i.e., $n = 99$) for *Min_slices*, and $k = 18$ and $r = 7$ (i.e., $n = 25$) for *Min_nodes*.

Our analysis confirms that, for a wide range of values for P_u and P_c and assumptions on the node availability p_u and compromise risk p_c, our approach is able to identify a configuration of r and k with manageable complexity (i.e., a reasonable number of replicas and of nodes). We note that, even when r and k grow, the minimum number of slices composing a resource remains limited.

VI. Implementation and Experiments

To verify the benefit of our proposal we applied it into an existing DCS network. Among the existing DCS networks (e.g., *Storj* [1], *Sia* [5], *IPFS* [4], and *Maidsafe Safe-network* [10]), we selected Storj since, to the best of our knowledge, it is currently the most advanced and supported DCS. The market valuation of the cryptocurrencies [11] associated with these DCSs (Storj for Storj, Siacoin for Sia, Filecoin for IPFS, and Maidsafecoin for Maidsafe) supports the importance that these solutions are rising: at the date of submission, the global market capitalization of these initiatives is more than 400 million dollars. There are currently more than 100,000 nodes offering capacity in the Storj network, with more than 100PB of data available and a planned goal of 10 times growth in 2019.

Storj is a protocol that coordinates a decentralized network to create and enforce storage contracts between peers. Each peer can negotiate contracts with other peers, upload and download data from other peers, and periodically verify the availability and integrity of her data. Storj leverages a Distributed Hash Table (DHT) to connect parties interested in forming a storage contract. In the discussion, we maintain the

terminology of our model and refer to parties outsourcing their resources to the decentralized network as owners (*renters* in Storj), and to parties offering storage space in exchange for a remuneration in a digital currency as storage nodes (*farmers* in Storj). *Bridge nodes* support the correct operations in the system and can take responsibility for the verification of the integrity and availability of resources. In the following, we describe the technical choices characterizing our implementation (Section VI-A), the experimental results (Section VI-B), and a few considerations about the impact produced by fine granularity retrieval (Section VI-C).

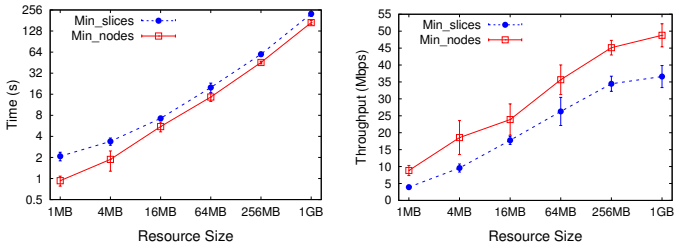


Fig. 9. Completion time (a) and overall throughput (b) in the *Min_slices* and *Min_nodes* allocation strategies

A. Implementation

The enforcement of *Min_slices* and *Min_nodes* allocation strategies in Storj required changing the client library of the open source implementation. In particular, Storj currently offers three main clients, one written in C that must be built from source, one written in JavaScript and designed to be executed by a node.js runtime, and one written in Python and compatible with any Python environment. We integrated our technique within the Python implementation, also for easy integration with the implementation of Mix&Slice, which in addition of being an AONT-encryption supports other protection requirements (e.g., encryption-based access control and policy revocation). The design of Storj makes the client independent from the bridge and the storage nodes. Our work on the Python client allowed us to access the services of the whole network.

We implemented the *Min_slices* and *Min_nodes* allocation strategies in the client and assigned slices (in the Storj terminology all the slices allocated to a node form a *shard*) to nodes. The performance of shard creation and resource reconstruction is orders of magnitude greater than the throughput of storage nodes in the Storj network (Mix&Slice operates at several hundred MB/s, whereas the maximum throughput we observed in Storj is around two orders of magnitude lower). In our experiments, we focused on evaluating the time required to complete the access request since the time requested by decryption does not have significant impact. We note that the use of the AONT forces each access request to be able to proceed with a client-side decryption only when the complete resource is available on the client. Should this be a problem for the specific application domain (e.g., resources are very large), mitigation can be provided by splitting the large resource and applying our approach to the resulting (smaller) chunks. Each chunk can then be downloaded and decrypted independently. This would reduce the access times to resources, but it may also delay the completion of the transfer, because the overhead for the management of a greater number of access requests reduces the effective bandwidth. The experiments confirm this observation (see Section VI-B), as they show that there is a performance benefit in managing large resources.

B. Experimental results

To evaluate the performance of the *Min_slices* and *Min_nodes* allocation strategies, we introduced into the client a module that activates a number of parallel threads (in the considered configuration, we used 10 concurrent threads) to open access requests to the storage nodes. In Storj, access requests from the owner involve both storage nodes and bridge nodes. In fact, each time an owner needs to retrieve a shard, she makes a request to the bridge, which returns a token together with the IP address of the node storing the required shard (note that this access request is recorded and a cryptocurrency payment is created by the owner for the node). The token is then used by the client as a parameter of an HTTP request directed to the node. Our experiments considered the performance of the system in the management of the dialogue between owner and node. In particular, we compared the access times observed for the two allocation strategies, varying the resource size.

An important restriction of the current implementation of Storj is that requests for shards are atomic and it is not possible to access only a specific portion of a shard managed by a node. This restriction cannot be removed operating only on the client, as it has a great impact not only on storage nodes but on the overall structure of the system. We then implemented the access requests for the *Min_slices* and *Min_nodes* techniques as follows. For *Min_nodes*, we implemented concurrent requests to the nodes. As soon as a node completes the delivery of its shard, a new request is started for another shard. The request is considered completed as soon as the client has received $k+1$ complete shards. For *Min_slices*, for each shard a number t of parallel threads ($t \leq r$) are activated to manage a request to distinct nodes managing the same shard (which coincides with a slice for this allocation strategy), for a number of shards compatible with the number of concurrent threads (e.g., in the experiments we set $t = 2$ and we had 5 shards processed at the same time by the 10 threads). As soon as a shard is fully delivered to the client, the group of t threads is dedicated to another missing shard.

Figure 9 reports the results of our experiments, where we used resources of size varying from 1MB to 1GB. Figure 9(a) shows the time required for the completion of the access requests and Figure 9(b) shows the throughput in terms of bandwidth. The graphs include two curves, one for *Min_slices* allocation, with $k = 26$ and $r = 4$, and one for *Min_nodes* allocation, with $k = 12$ and $r = 5$, which correspond to the configurations considered in Figure 8(a) and Figure 8(b), respectively. The graphs present the average and the standard deviation of the values obtained with 10 executions. We note that the *Min_nodes* strategy exhibits a moderate ben-

efit compared to the *Min_slices* strategy. The benefit derives from the savings in overhead associated with the interaction with multiple nodes. An element that also contributes to the throughput is the natural variety of performance in nodes, with some being faster than others. The *Min_nodes* strategy works well as long as the number of nodes with limited performance (*slow* nodes) is less than $r-1$ and there are at least $k+1$ nodes with good performance (*fast* nodes) serving the shard. For *Min_slices*, it is sufficient to have one of the $k+1$ shards assigned to a group of nodes where the t nodes contacted in parallel happen to be slow to suffer from a significant delay in the access.

C. Further considerations

We noted that a limitation of the current implementation of the Storj node is that the request for a shard is atomic. The realization of a mechanism that permits to manage partial access requests would offer the opportunity for a significant improvement in the management of the access requests. For a generic server in a file sharing protocol this can be expected to be a relatively simple change in the server code; for a DCS system, this change would require a revision of the model used for the remuneration of access requests, as follows. The bridge should not consider each request received by the owner as an access to the complete resource (which also implies a payment to access the whole resource), but it should return to the owner the IP address and the authorization token of the node storing the shard. The owner and the node should then commence a protocol in which the owner issues signed confirmations in exchange of pieces of the resource. These confirmations can then be submitted to the bridge to receive the payment. Allowing the owner to pay a node only for the downloaded portion of the resource results in better performance and stronger competition among the nodes; best performing nodes would be preferred by the owner and would serve more traffic, receiving a correspondingly greater remuneration for their storage service.

The flexible structure of the *Min_nodes* assignment would be particularly suited to this model. Under the assumption that nodes exhibit a high variability in access times, each slice could be retrieved by any of the r nodes storing it, with the possibility to adapt the amount of data transferred from each node depending on the response time and in case a group of r nodes happens to be all composed of slow nodes, the impact would be limited to the single or few slices that cannot be retrieved from fast nodes.

VII. Related Work

RAID [12] is one of the main contributions aimed at the construction of reliable systems. RAID is normally deployed on local drives. With the advent of the cloud, RAID has been extended to take adversarial failures into consideration. Along this line of works, *HAIL (High-Availability and Integrity Layer)* [13] extended RAID with multiple cloud storage providers and a *Proof of Retrievability (PoR)* [14] scheme to verify that a provider still holds a certain piece of information. *HAIL* is however not well-suited for DCS systems, where the nodes are less reliable than well-established cloud service providers. Also, *HAIL* does not take into account the possibility of adversarial users trying to reconstruct the resources for their own personal profit. The works closest to ours are the solutions aimed to offer reliability and security of data in DCS. Many DCS networks that have recently been proposed, already include a certain degree of security guarantees. (i.e., protection against malicious parties jointly collecting all the slices composing a resource). Among them, Storj [1] and Sia [5] adopt client-side encryption and do not protect the outsourced data against coalitions of malicious nodes. SAFE Network [2] instead adopts a self-encryption technique: the resource is divided into shards and a weak AONT among 3 shards is applied before uploading them. In [3] the design of the SAFE Network and the possible attack vectors are analyzed. The solution proposed in [2], [3] is predetermined and the interaction between redundancy and security is not analyzed. Our proposal could be applied to improve the flexibility and security of these networks.

Another line of works is security of outsourced data (e.g., [15], [16], [17], [18]), which can be improved using AONT. Existing solutions however consider domains different from DCS. We have discussed before the proposal in [9], where the goal was to support policy evolution for outsourced resources where the access control policy is mapped to an encryption policy. Another approach using AONT and Reed-Solomon codes is AONT-RS [19]. Apart from the use of AONT, there is a limited similarity with the structure of our proposal. In fact, the work in [19] does not explicitly consider the structure of current DCS systems and does not provide an approach for the identification of the parameters to use in the configuration of the system. An evolution of the work on AONT-RS is CAONT-RS [20] that has been used by CDStore [21], which also uses two-stage deduplication to achieve both bandwidth and storage savings and robustness against side-channel attacks, while DepSky [22] addresses the privacy requirements using Shamir's scheme. All these proposals consider cloud-of-clouds environments, which see the integration of the services of cloud providers. Their adaptation to the DCS scenario requires significant attention and a model for the identification of the parameters to use in the configuration of the system. Also, the interaction between security and availability is not analyzed.

A precursor of DCS is represented by P2P systems. The P2P system closer to our proposal, which considers reliability and security, is Tangler [23]. The goal of Tangler is censorship resistance, which is a potential application of DCS, but not its main goal. Several of the assumptions at the basis of the design of Tangler have also been considered in the realization of DCS systems. A crucial difference between Tangler and our proposal is that Tangler uses Shamir's method, so it is quite expensive in terms of storage and bandwidth. Also, it does not aim at combining availability and confidentiality requirements in data allocation.

The novelty of our approach with respect to all above-mentioned techniques is the combination of AONT with different strategies for slicing and allocating resources in DCS systems and the joint consideration of security and availability

guarantees. Our analysis of the characterization, interplay, and settings of the parameters guiding slicing and allocation can be used by all existing solutions to enhance their security and availability properties.

VIII. Conclusions

We presented an approach for providing effective secure protection to resources in decentralized cloud storage services. Our approach enables resource owners to protect their resources and to control their decentralized allocation to different nodes in the network. We investigated different strategies for splitting and distributing resources, analyzing their characteristics in terms of availability and security guarantees. We also provided a modeling of the problem enabling owners to control the granularity of slicing and diversification of allocation to ensure aimed availability and security guarantees. Enabling effective control for resource owners, our solution helps in removing natural reluctance due to security concerns and moves a step forward in the realization of novel services effectively benefiting from technological evolution. Our work leaves room for extensions, such as the consideration of error correcting codes and information dispersal algorithms to reduce the spatial overhead.

Appendix A
Proofs of Theorems

Proof of Theorem 1:
Let us assume, by contradiction, the existence of a (k,r)-allocation for a resource split into $s = k < k+1$ slices. Given s different slices, no more than s nodes can be used to store one replica of the slices. Since s=k, the allocation function is storing the whole resource using at most k nodes. Therefore, it is not k-protected. Note that $k+1$ slices are sufficient to define a (k,r)-allocation. The r-replication requirement is easily satisfied by replicating r times each of the $k+1$ slices. The k-protection requirement is satisfied by storing each slice to a different node. Hence, for each replica of the resource, each coalition of k nodes misses one slice (the one stored at the $(k+1)$-th node). ■

Proof of Theorem 2:
Since there are $r\cdot(k+1)$ slices to be stored, a (k,r)-allocation cannot use more than $r\cdot(k+1)$ nodes. However, a (k,r)-allocation with $s = k+1$ cannot use less than $r\cdot(k+1)$ nodes. Let us consider the case where slices are not replicated (i.e., $r = 1$), since the same discussion applies to each replica of the resource. Assume, by contradiction, that a (k,r)-allocation stores more than one slice at one of the nodes. If the function adopts $n = k+1$ nodes, there will be at least one node that does not store any slice (which is equivalent to say that $n \leq k$) as the number of slices is $k+1$. Then, k nodes store all the slices composing the resource, thus violating k-protection. ■

Proof of Theorem 3:
To guarantee r-replication, each slice should be stored at (at least) r nodes. If allocation function φ is k-protected, for each coalition $\mathtt{N_i}$ of k nodes, there exists at least a slice $\mathtt{s_j}$ that is not stored at any of the nodes in $\mathtt{N_i}$. To guarantee that $\mathtt{s_j}$ has r copies, there must exist at least r additional nodes that store $\mathtt{s_j}$, and n should be at least equal to $k+r$. Note that $k+r$ nodes are sufficient to define a (k,r)-allocation. Consider, as an example, $s = \binom{k+r}{k}$ slices, the set $\mathtt{N_1},\ldots,\mathtt{N_s}$ of possible coalitions of k nodes, and allocation function φ that assigns the i-th slice to all the nodes in $\mathcal{N}$, but the ones in the i-th coalition: $\varphi(\mathtt{s_i}) = \mathcal{N} \setminus \{\mathtt{N_i}\}$. Function φ is k-protected, since each coalition $\mathtt{N_i}$ cannot access slice $\mathtt{s_i}$, and r-replicated, since slice $\mathtt{s_i}$ is stored at each node $\mathtt{n_i} \in \mathcal{N} \setminus \{\mathtt{N_i}\}$, then at $n-k = r$ nodes. ■

Proof of Theorem 4:
An allocation function φ is k-protected if the set of slices stored at any coalition $\mathtt{N_i}$ of k nodes is not complete (i.e., at least one slice is missing). To guarantee that φ is an allocation function each slice should be stored at least on one node. If each coalition misses a different slice, we are minimizing the number of slices necessary to define an allocation function φ. Indeed, since $n > k$, the missing slice for each coalition $\mathtt{N_i}$ can be stored at the nodes in $\mathcal{N} \setminus \mathtt{N_i}$, as they will miss another slice. Since there are $\binom{k+r}{k}$ possible coalitions of k nodes in $\mathcal{N}$, s should be at least $\binom{k+r}{k}$ to guarantee that each coalition misses a different slice. Assume, by contradiction, that $r = 1$, $s = \binom{k+r}{k} - 1$, and that φ is k-protected. In this case, two coalitions $\mathtt{N_i}$ and $\mathtt{N_j}$ will miss the same slice $\mathtt{s_x}$. That is, there are at least $k+1$ nodes (the ones in $\mathtt{N_i} \cup \mathtt{N_j}$) missing $\mathtt{s_x}$. However, if $n = k+1$ then φ cannot be an allocation function, since no node in $\mathcal{N}$ stores $\mathtt{s_x}$. (The same reasoning applies with larger values for r.) ■

Proof of Theorem 5:
1) Since each coalition of k nodes should not be able to reconstruct the resource, it should miss at least one slice. The number of slices used by the allocation function is $s = \binom{k+r}{k}$, which is sufficient for each coalition to miss at least one slice. In fact, the number of possible coalitions of k nodes is $\binom{k+r}{k}$. Let us assume, by contradiction, that two coalitions $\mathtt{N_i}$ and $\mathtt{N_j}$ miss the same slice $\mathtt{s_x}$. Therefore, there are $k+1$ nodes ($\mathtt{N_i} \cup \mathtt{N_j}$) that do not store $\mathtt{s_x}$. However, in this case $\mathtt{s_x}$ would be stored at $n-(k+1) = r-1$ nodes. Hence, the allocation function would not be r-replicated.

2) By definition of (k,r)-allocation with $n = k+r$ nodes and $s = \binom{n}{k}$ slices, each coalition of k nodes misses a different slice. Let us consider two coalitions $\mathtt{N_i}$ and $\mathtt{N_j}$ that differ in one node only. Coalition $\mathtt{N_j}$ misses one slice, $\mathtt{s_j}$, while $\mathtt{N_i}$ misses $\mathtt{s_i}$. Since $\mathtt{N_j}$ misses one slice only, it stores $\mathtt{s_i}$. Hence, $\mathtt{N_i} \cup \mathtt{N_j}$ includes $k+1$ nodes and stores all the slices composing the resource. ■

Proof of Theorem 6:
P_u) The probability to obtain back the original plaintext resource corresponds to the probability that $k+1$ nodes, each storing a different slice, do not fail. Since p_u is the probability that a node fails, the probability that at least one of the r replicas of a slice is available is $(1-(p_u)^r)$, with $(p_u)^r$ the probability that all r replicas of the slice are unavailable. The probability to obtain back all the $k+1$ slices, and then also the original plaintext resource, is then $(1-(p_u)^r)^{k+1}$ and $(1-(1-(p_u)^r)^{k+1})$ is the probability that the resource cannot be decrypted.

P_c) The probability that the resource is exposed is the proba-

bility that all the slices are compromised, meaning that $k+1$ nodes are malicious. Since $1-p_c$ is the probability that a node is not compromised and each slice has r replicas, the probability that at least one replica is exposed is $(1-(1-p_c)^r)$, with $(1-p_c)^r$ the probability that all r replicas of a slice are not compromised. We can then conclude that the probability that all $k+1$ slices are exposed is $(1-(1-p_c)^r)^{k+1}$. ■

Proof of Theorem 7:

P_u) To obtain back the original plaintext resource, we need the slices stored on any combination of $k+1$ nodes, that is, $k+1$ nodes must not fail. A resource therefore becomes unavailable when any combination of r or more nodes fail. The probability that i nodes fail, with $i = r, \ldots, k+r$, and $k+r-i$ nodes do not fail is equal to $(p_u)^i(1-p_u)^{k+r-i}$. Since the number of combinations of i nodes out of $k+r$ is $\binom{k+r}{i}$, the probability that a resource is unavailable is $P_u = \sum_{i=r}^{k+r} \binom{k+r}{i}(p_u)^i(1-p_u)^{k+r-i}$.

P_c) A coalition can compromise the confidentiality of the resource whenever it involves any combination of $k+1$ nodes, that is, at least $k+1$ nodes must be compromised. The probability that i nodes are compromised, with $i = k+1, \ldots, k+r$, and $k+r-i$ nodes are not compromised is equal to $(p_c)^i(1-p_c)^{k+r-i}$. Since the number of combinations of i nodes out of $k+r$ is the binomial coefficient $\binom{k+r}{i}$, we can conclude that the probability that any combination of at least $k+1$ nodes are compromised in a collection of $r+k$ nodes is $P_c = \sum_{i=k+1}^{k+r} \binom{k+r}{i}(p_c)^i(1-p_c)^{k+r-i}$. ■

ACKNOWLEDGMENTS

This work was supported in part by the EC within the H2020 Program under grant agreement 825333 (MOSAICrOWN).

REFERENCES

[1] S. Wilkinson, T. Boshevski, J. Brandoff, J. Prestwich, G. Hall, P. Gerbes, P. Hutchins, C. Pollard, and V. Buterin, "Storj: a peer-to-peer cloud storage network (v2.0)," https://storj.io/storjv2.pdf, Storj Labs Inc., Tech. Rep., 2016.

[2] D. Irvine, "Maidsafe distributed file system," MaidSafe, Tech. Rep., 2010.

[3] G. Paul, F. Hutchison, and J. Irvine, "Security of the maidsafe vault network," in *Wireless World Research Forum Meeting 32*, Marrakesh, Morocco, May 2014.

[4] J. Benet, "IPFS-content addressed, versioned, P2P file system," Protocol Labs, Tech. Rep., 2014.

[5] D. Vorick and L. Champine, "Sia: Simple decentralized storage," https://sia.tech/sia.pdf, Nebulous Inc., Tech. Rep., 2014.

[6] C. Patterson, "Distributed content delivery and cloud storage," https://www.smithandcrown.com/distributed-content-delivery-cloud-storage/, Smith and Crown, Tech. Rep., 2017.

[7] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra, "Executing SQL over encrypted data in the database-service-provider model," in *Proc. of ACM SIGMOD*, Madison, Wisconsin, June 2002.

[8] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, September/December 1979.

[9] E. Bacis, S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, M. Rosa, and P. Samarati, "Mix&Slice: Efficient access revocation in the cloud," in *Proc. of ACM CCS*, Vienna, Austria, October 2016.

[10] N. Lambert and B. Bollen, "The SAFE network - a new, decentralised internet," http://docs.maidsafe.net/Whitepapers/pdf/TheSafeNetwork.pdf, MaidSafe, Tech. Rep., 2014.

[11] M. Conti, E. S. Kumar, C. Lal, and S. Ruj, "A survey on security and privacy issues of bitcoin," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3416–3452, 2018.

[12] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," *ACM SIGMOD Records*, vol. 17, no. 3, pp. 109–116, Jun. 1988.

[13] K. D. Bowers, A. Juels, and A. Oprea, "HAIL: A high-availability and integrity layer for cloud storage," in *Proc. of ACM CCS*, Chicago, IL, USA, November 2009.

[14] ——, "Proofs of retrievability: Theory and implementation," in *Proc. of ACM CCSW*, Chicago, IL, USA, November 2009.

[15] M. Albanese, S. Jajodia, R. Jhawar, and V. Piuri, "Dependable and resilient cloud computing," in *Proc. of IEEE SOSE*, Oxford, UK, March 2016.

[16] A. Aldribi, I. Traore, and G. Letourneau, "Cloud slicing a new architecture for cloud security monitoring," in *Proc. of IEEE PACRIM*, Victoria, Canada, August 2015.

[17] D. Nuñez, I. Agudo, and J. Lopez, "Delegated access for hadoop clusters in the cloud," in *Proc. of IEEE CloudCom*, Singapore, December 2014.

[18] M. Theoharidou, N. Papanikolaou, S. Pearson, and D. Gritzalis, "Privacy risk, security, accountability in the cloud," in *Proc. of IEEECloudCom*, Bristol, UK, December 2013.

[19] J. K. Resch and J. S. Plank, "AONT-RS: blending security and performance in dispersed storage systems," in *Proc of FAST*, San Jose, CA, USA, February 2011.

[20] M. Li, C. Qin, P. P. C. Lee, and J. Li, "Convergent dispersal: Toward storage-efficient security in a cloud-of-clouds," in *Proc. of HotStorage*, Philadelphia, PA, USA, June 2014.

[21] M. Li, C. Qin, and P. P. C. Lee, "CDStore: Toward reliable, secure, and cost-efficient cloud storage via convergent dispersal," in *Proc. of USENIX ATC*, Santa Clara, CA, USA, July 2015.

[22] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, "DepSky: Dependable and secure storage in a cloud-of-clouds," *ACM TOS*, vol. 9, no. 4, pp. 12:1–12:33, 2013.

[23] M. Waldman and D. Mazieres, "Tangler: a censorship-resistant publishing system based on document entanglements," in *Proc. of ACM CCS*, Philadelphia, PA, USA, November 2001.

Enrico Bacis — PhD Student, Università degli Studi di Bergamo, Italy. His work focuses on computer security, and in particular on investigating the integration of security and privacy features in mobile, cloud, and database systems. He has worked on security-related projects in Google Munich (DE) and Google London (UK). https://cs.unibg.it/bacis

Sabrina De Capitani di Vimercati — Professor, Università degli Studi di Milano, Italy. Her research interests are in data security and privacy. She has published more than 210 papers in journals, conference proceedings, and books. She has been a visiting researcher at SRI International, CA (USA), and George Mason University, VA (USA). She chairs the IFIP WG 11.3 on Data and Application Security and Privacy. http://www.di.unimi.it/decapita

Sara Foresti — Professor, Università degli Studi di Milano, Italy. Her research interests are in data security and privacy. She has published more than 100 papers in journals, conference proceedings, and books. She has been a visiting researcher at George Mason University, VA (USA). She has been serving as General chair and PC chair of several conferences. She chairs the IEEE CS Italy Chapter. http://www.di.unimi.it/foresti

Stefano Paraboschi — Professor, Università degli Studi di Bergamo, Italy. His research focuses on information security and privacy, Web technology for data intensive applications, XML, information systems, and database technology. He has been a visiting researcher at Stanford University and IBM Almaden, CA (USA), and George Mason University, VA (USA). https://cs.unibg.it/parabosc

Marco Rosa — PhD Student, Università degli Studi di Bergamo, Italy. His research interests focus on cloud and mobile security, and access control policies. He has been a visiting researcher at SAP Labs France. https://cs.unibg.it/rosa

Pierangela Samarati — Professor, Università degli Studi di Milano, Italy. Her main research interests are in data protection, security, and privacy. She has published more than 270 papers in journals, conference proceedings, and books. She has been a visiting researcher at Stanford University, CA (USA), SRI International, CA (USA), and George Mason University, VA (USA). She has been named ACM Distinguished Scientist (2009) and IEEE Fellow (2012). She has received the IEEE Computer Society Technical Achievement Award (2016). http://www.di.unimi.it/samarati