Dynamic Allocation for Resource Protection in Decentralized Cloud Storage

Enrico Bacis[*], Sabrina De Capitani di Vimercati[†], Sara Foresti[†],
Stefano Paraboschi[*], Marco Rosa[*‡], Pierangela Samarati[†]
[*] Università degli Studi di Bergamo, Italy – Email: *name.surname*@unibg.it
[†] Università degli Studi di Milano, Italy – Email: *name.surname*@unimi.it
[‡] Security Research, SAP Labs France

***Abstract*—Decentralized Cloud Storage (DCS) networks represent an interesting solution for data storage and management. DCS networks rely on the voluntary effort of a considerable number of (possibly untrusted) nodes, which may dynamically join and leave the network at any time. To profitably rely on DCS for data storage, data owners therefore need solutions that guarantee confidentiality and availability of their data. In this paper, we present an approach enabling data owners to keep data confidentiality and availability under control, limiting the owners intervention with corrective actions when availability or confidentiality is at risk. Our approach is based on the combined adoption of AONT (All-Or-Nothing-Transform) and fountain codes. It provides confidentiality of outsourced data also against malicious coalitions of nodes, and guarantees data availability even in case of node failures. Our experimental evaluation clearly shows the benefits of using fountain codes with respect to other approaches adopted by current DCS networks.**

I. Introduction

Information technology has seen considerable advancements at a very fast pace in recent years. A common aspect in the technological evolution at support of information management has been the emergence of services enabling data owners to rely on external parties for the storage of their data. For a few years now, the cloud has become the reference paradigm for the storage and management of data, providing data owners with scalable resources at profitable prices. As a matter of fact, relying on Cloud Service Providers (CSPs), data owners can acquire resources for storing their data as needed and with considerable economic benefits with respect to in-house solutions providing the same quality of service. While the cloud paradigm typically is based on a single service provider, the emergence of blockchains and micro-payment networks has introduced a new interest towards Decentralized Cloud Storage (DCS) networks. DCS networks rely on storage capabilities of network peers and provide an interesting emerging alternative to traditional cloud storage. The main characteristic of a DCS is that storage space is provided by users, which can be anyone from big providers to individuals, that can join the network and offer storage space, typically in exchange of some reward (micro-payment). Examples of DCS systems are Storj [17], SAFE Network Vault [11], [7], IPFS [5], [10], and Sia [18], which enable users to rent out their unused storage and bandwidth in exchange of some crypto-currency. The great advantage is that the price for storage on a DCS is a fraction of the price of a normal provider.

The use of a dynamic network of unknown (and potentially non-trusted) peers clearly introduces the problem of guaranteeing proper protection to resources, ensuring their availability and security (for both confidentiality and integrity). In fact, a DCS is a potentially unstable network, and hence continuous participation of every single node cannot be assumed. Given this, and in line with the distributed nature of DCS services, resources are sliced into many shards, with different shards allocated to different nodes of the network, with replication to guarantee availability. Reconstruction of a resource requires collecting the different shards composing it. Also, nodes participating in the DCS - which can dynamically join and leave and are anonymous - cannot be considered trusted, hence resource confidentiality needs to be protected against each of them (as well as against possible coalitions) and owners should be able to assess integrity of the shards (and hence resources).

Typically, DCS networks provide security by encrypting the resource at the owner side before slicing it, and provide availability by employing shard replication. Instead of simple replication in the allocation, several DCS networks leverage the dynamic application of erasure codes (e.g., Reed-Solomon). With Reed-Solomon, if a node participating in the service becomes unavailable, its shard is dynamically re-allocated to another node. The advantage of Reed-Solomon with respect to simple replication is that it provides reliability guarantees at a fraction of the storage overhead that would come with replication. However, it has two main drawbacks. First, it is a fixed-rate encoding technique and therefore computing the shard to be re-allocated requires reconstructing the complete resource. Second, if the old node originally storing a (then re-allocated) shard comes back online, more replicas than actually needed would be available for the same shard and the economic cost brought by the involvement of more nodes does not bring a clear advantage for availability. Also, the use of simple encryption for providing confidentiality has some limitations since, although it is true that the unavailability of all shards composing a resource prevents its complete reconstruction, still a subset of the shards may enable reconstruction of a portion of the resource to users knowing the encryption key. For instance, a owner losing control on the encryption key

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

may ask nodes in the DCS to delete the resource, as it would otherwise remain exposed. However, nodes that do not respect the request of deleting their shards may then have access to a portion of the resource.

The contribution of this paper is twofold. First, we combine All-or-Nothing-Transform (AONT), in contrast to simple encryption for protecting resources, and fountain codes, in contrast to traditional erasure codes like Reed-Solomon for computing shards to be distributed to nodes in a DCS (Section III). The combined adoption of AONT and fountain codes provides better security and availability guarantees, and lower performance overhead than current solutions. Second, we provide a model for dynamically managing the computation and allocation of shards, avoiding an immediate reaction to a node leaving, if availability guaranteed by the remaining nodes is considered still acceptable. Our approach avoids potentially excessive generation of shards which may turn out to be unnecessary when - as it is often the case - nodes unavailability is only temporary (Section IV). Avoidance of excessive generation provides advantages not only for performance (as it avoids generating new shards and involvement of new nodes) but also for security (excessively increasing the number of shards in the system may cause a higher vulnerability to malicious coalitions). The advantage of our solution is confirmed by experimental results obtained by integrating our approach with a real DCS (Section V).

II. Basic concepts

The two basic building blocks enabling the development of our solution are the adoption, at the client side, of *All-or-Nothing-Transform* (AONT) and of *fountain codes*.

AONT. AONT [15] is an encryption mode that transforms a plaintext resource into a ciphertext where every bit of the output has strong interdependence on all the bits of the input (e.g., [2], [8], [9]). In other words, AONT ensures that even knowing the encryption key, a resource or a portion of it cannot be reconstructed even when even only a single block is missing. AONT then provides better security guarantees than encryption.

Fountain codes. Fountain codes are a class of erasure codes preventing that the loss of one of the transmitted or stored blocks of a resource causes a data loss. Given a resource r, partitioned into f different *fragments*, an erasure code generates a set of $s>f$ *encoded shards* that depend on the resource content and support the reconstruction of r through the combination of a subset of the encoded shards. Fountain codes, unlike other erasure codes (e.g., Reed-Solomon [13]), offer probabilistic reconstruction guarantees, meaning that with a probability $p<1$, f of the s shards are sufficient for reconstructing r. The reconstruction probability p exponentially increases by retrieving additional shards. Although probabilistic, fountain codes have two main characteristics that, as we will discuss in the next section, allow us to profitably use them in the DCS context. First, they are *rateless*, that is, using these codes it is possible to create a new (i.e., different from each other) shard on the fly and therefore the number s of encoded shards is not fixed a priori. Independently from the number s of shards, any subset of (at least) f shards can be used to reconstruct r. Second, each shard depends on a subset of (and not on all) the f original fragments of the resource and then only a subset of the original fragments are needed for generating a new shard.

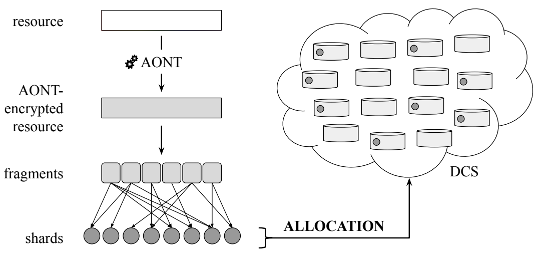


Fig. 1: Reference scenario

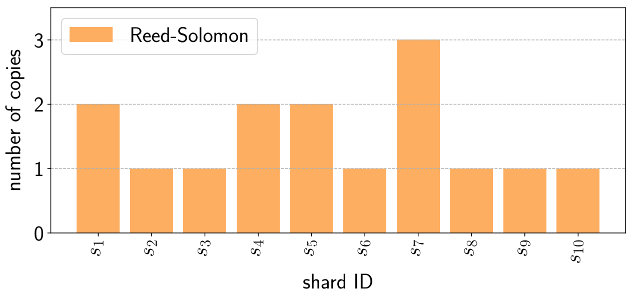
III. Encoding and allocation strategy

We consider a data owner interested in storing her resources in a DCS network. For simplicity, we refer the discussion to a single resource r, since the same approach can be applied to all the resources moved to the DCS.

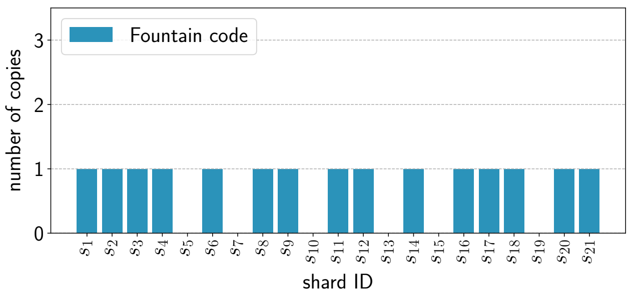
Given a resources r we assume the data owner to first encrypt r using AONT, to protect confidentiality, and then encode the resulting ciphertext using fountain codes, to provide availability. As illustrated in Figure 1, the ciphertext is organized in f original fragments and encoded into a set $\mathcal{S}=\{\mathtt{s_1},\ldots,\mathtt{s_s}\}$ of s shards allocated to s randomly chosen nodes $\mathcal{N}=\{\mathtt{n_1},\ldots,\mathtt{n_s}\}$ (i.e., each shard is allocated to a different node). Whenever a user is interested in retrieving the resource content, she contacts an arbitrary subset of f nodes and downloads their shards, which are then combined to reconstruct the resource.

The reason for using AONT, in contrast to encryption, is its ability to prevent even partial reconstruction of the plaintext resource. This property is crucial in the considered scenario, since fountain codes enable the reconstruction of a fragment (i.e., a chunk of the ciphertext) starting from a small subset of shards (i.e., less than f shards). The adoption of AONT then prevents any (small) coalition of malicious nodes to partially reconstruct the plaintext resource content, even if they can reconstruct a fragment from the shards they store.

The choice of using fountain codes, in contrast to simple replication or traditional erasure codes (e.g., Reed-Solomon), is due to their high flexibility in dealing with the intrinsically dynamic behavior of DCS . Since fountain codes are rateless, if a node $\mathtt{n_i}$ leaves the DCS, the data owner does not need (as it would happen using Reed-Solomon) to reconstruct its shard $\mathtt{s_i}$ and re-allocate it to a different node to guarantee the same resource availability. Indeed, it is sufficient to generate a new shard $\mathtt{s_{s+1}}$ (different from $\mathtt{s_1},\ldots,\mathtt{s_s}$) and allocate it to a new node $\mathtt{n_{s+1}}$. After the generation and allocation of $\mathtt{s_{s+1}}$, any user can still reconstruct the resource content by downloading any subset of f shards from the resulting set $(\mathcal{S}\backslash\{\mathtt{s_i}\})\cup\{\mathtt{s_{s+1}}\}$ of s shards. The generation of a new shard,



(a) Reed-Solomon



(b) Fountain codes

Fig. 2: An example of shard generation/replication using Reed-Solomon (a) and fountain codes (b) in a dynamic scenario

in contrast to the replication of the unavailable one, provides higher resource availability. Indeed, every shard is unique and equally contributes to the reconstruction of the resource. Hence, when the previously unavailable node $\mathtt{n_i}$ comes back online, the increased economic cost due to the involvement of a larger number $(s+1)$ of nodes comes with an actual advantage in terms of higher availability. In fact, there will be $s+1$ (instead of s) different shards stored at $s+1$ different nodes that can all be used for resource reconstruction. Note that the generation of a new shard causes a limited overhead since it implies the download of a subset of the shards in $\mathcal{S}$, without the need to reconstruct r (as required by other erasure codes).

As discussed, the rateless characteristic of fountain codes allows the adaptive adjustment of the number of shards available for reconstructing a resource, thus impacting the availability and security guarantees. We use this characteristic (see Section IV) in such a way that, when the availability guarantees go below a given threshold, the owner can generate a new shard. Analogously, when the risk of confidentiality exposure is above a given threshold due to the presence of a high number of shards, the data owner can re-encrypt the resource and generate a new set of s shards.

Figure 2 illustrates an example that compares the set of shards in a DCS when using Reed-Solomon and fountain codes. Here, we assume that the data owner partitions the resource in f=7 fragments and encodes it in s=10 shards and that nodes where the shards are stored leave and then possibly re-join the network. Since Reed-Solomon reacts to a node failure replicating the shard of the failed node, the set $\mathcal{S}$ of shards representing r never changes but the number of copies grows (e.g., $\mathtt{s_7}$ has three copies). Fountain codes do not cause any replication, but a new shard is generated at each failure. Note that the two techniques imply the same economic cost for the owner, since each failure causes the allocation of a (new or duplicated) shard to a node. In the considered example, the owner pays for 15 shards in both scenarios.

IV. Availability and Security Guarantees

Node failure has an impact on both resource availability and confidentiality. Indeed, when one of the nodes $\mathtt{n_i}$ fails, the encoded shard $\mathtt{s_i}$ it stores cannot be used to reconstruct the resource content. Hence, the user needs to retrieve f out of $s-1$, in contrast to s, shards. When $\mathtt{n_i}$ re-joins the DCS, it still stores $\mathtt{s_i}$ and this could provide a positive effect on resource availability, since the shard at the node could be used to reconstruct the resource. However, node $\mathtt{n_i}$ re-joining the DCS could have a negative impact on security, since $\mathtt{n_i}$ could exploit its knowledge of $\mathtt{s_i}$ and collude with other $f-1$ nodes to reconstruct r (or prevent its deletion). Similarly, the generation and allocation of a new shard $\mathtt{s_{s+1}}$ improves resource availability, but also naturally reduces security since there is a higher number of nodes that could possibly collude.

In this section, we analyze the advantages provided by fountain codes on availability guarantees, by studying the probability P_u that a resource becomes unavailable as a consequence of nodes leaving and re-joining the network. We also evaluate the risks that the adoption of our solution causes in terms of security, by studying the probability P_c that a coalition of malicious nodes has enough shards to compromise resource security. These probabilities depend on the probability p_u that a node fails, and on the probability p_c that a node is malicious and interested in colluding with other nodes to breach the confidentiality of the resource. For simplicity, we assume p_u and p_c to be the same for all the nodes.

A. Availability guarantees

When using s shards to encode a resource r split into f original fragments, r becomes unavailable when more than $s-f$ nodes fail (i.e., when less than f shards can be accessed). The probability of such an event to happen is P_u=$\sum_{i=s-f+1}^{s}\binom{s}{i}{p_u}^i(1-p_u)^{s-i}$. Probability P_u increases when one of the nodes fails (or leaves the DCS) since the shard it stores is no more available, while it decreases any time a new shard is generated or a failed node re-joins the network.

Even if, in principle, the owner should react every time a node $\mathtt{n_i}$ fails by generating a new shard, this practice is expensive and may not even be necessary (e.g., if $\mathtt{n_i}$ re-joins the network in a few hours). The increase in P_u caused by the failure of a node may not be critical in a scenario where nodes dynamically leave and re-join the system frequently. Indeed, the reduction in P_u may be temporary and may not

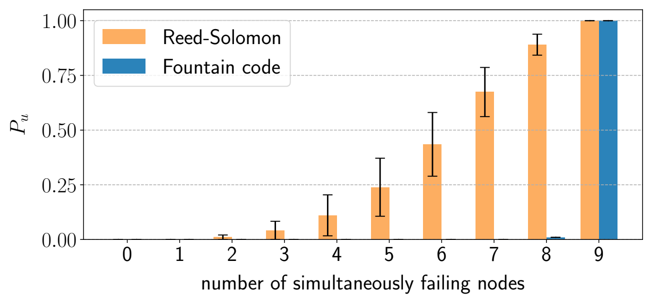


Fig. 3: Probability that a resource becomes unavailable using Reed-Solomon and fountain codes, assuming p_u=0.015, P_u^{min}=10^{-12}, P_u^{max}=10^{-5}, f=7 fragments, and s in [10,15]

considerably affect the ability of the user to reconstruct r. To properly take into consideration these aspects, we propose a solution where the data owner takes corrective actions (i.e., create a new shard) only when resource availability is considered at risk.

Our solution is based on the definition of two thresholds for P_u, P_u^{min} and P_u^{max}, identifying the range of values considered acceptable by the owner for guaranteeing the availability of r. Intuitively, these thresholds influence the maximum and minimum number of available shards in the system and represent:

- P_u^{max}: the maximum probability of failure that the owner can tolerate, which corresponds to the minimum number of shards ($\geq f$) the owner considers desirable to keep resource unavailability under control;
- P_u^{min}: the minimum probability of failure fixed by the data owner based on her economic availability, which corresponds to the maximum number of shards the owner can afford.

The data owner does not react every time a node leaves or re-joins the DCS, but only if this event causes P_u to become higher than P_u^{max} or smaller than P_u^{min}. If P_u exceeds P_u^{max}, the data owner generates a new shard and allocates it to a new node. If P_u goes below P_u^{min}, the data owner can terminate the contract with one of the nodes in the system (e.g., with the one that has been off-line the most) and stop paying for its services.

Consider, as an example, a resource partitioned in f=7 original fragments and encoded in s=10 shards allocated at nodes with probability p_u=0.015 of failure. Figure 3 compares the probability P_u of unavailability of a resource r when using Reed-Solomon and fountain codes, assuming P_u^{min}=10^{-12} and P_u^{max}=10^{-5}, and varying the number of nodes that the data owner identifies as unavailable between 0 and 9. The error bars in the figure represent the standard deviation. Note that P_u is initially the same for Reed-Solomon and fountain codes, and evolves in the same manner when nodes leave the DCS. The evolution of P_u when a node re-joins the DCS is instead considerably different: with Reed-Solomon it causes duplication of a shard, with fountain codes it implies the

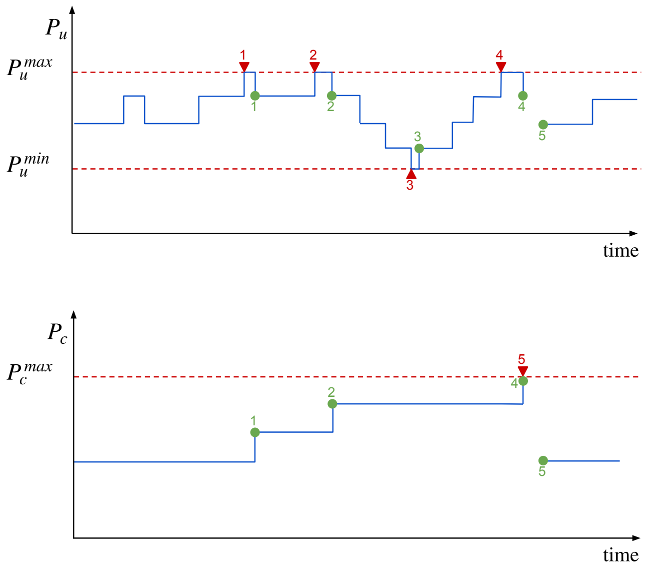


Fig. 4: An example of evolution of P_u (top chart) and P_c (bottom chart) as a consequence of nodes leaving and re-joining the DCS and of actions taken by the data owner

availability of an additional (different) shard. As visible from the figure, the probability that r becomes unavailable is higher using Reed-Solomon than using fountain codes. In fact, the nodes storing shards available in a single copy (e.g., s_2 in Figure 2(a)) play a critical role in resource reconstruction. Indeed, when failing, they cannot be immediately substituted. With fountain codes, all nodes are equally critical since they all store different shards that can be interchangeably used to reconstruct resource r.

The top chart in Figure 4 illustrates an example of evolution of P_u, assuming the adoption of fountain codes, considering the corrective actions taken by the owner. The value of P_u grows every time a node leaves and decreases every time a node re-joins the network. As long as P_u is between P_u^{min} and P_u^{max} (the two red dashed lines in the figure), the data owner does not react to node leave and re-join events. In the example, we set P_u^{min} and P_u^{max} in such a way to tolerate the leave and re-join of one node at a time. When P_u reaches P_u^{max} as a consequence of the failure of a node (red triangles 1, 2, and 4 in the figure, indicating unavailability of two nodes), the owner generates and allocates a new shard. The generation of a new shard reduces P_u to a value below P_u^{max} (green circle 1, 2, and 4 in the figure). When P_u reaches P_u^{min} as a consequence of node re-join (red triangle 3 in the figure, indicating re-join of all the three nodes that failed), the owner closes the contract with one of the nodes and P_u returns above the threshold (green circle 3 in the figure).

In the next section, we will illustrate that P_u decreases not only when the data owner creates a new shard, but also when the resource is re-encrypted.

B. Security guarantees against malicious coalitions

Since f shards are sufficient to reconstruct r, any coalition of at least f malicious nodes can reconstruct the encrypted content of the resource (and its plaintext representation if the key is exposed) and/or prevent its deletion by retaining their local copy of the shard. The probability that f (or more) nodes collude is $P_c = \sum_{i=f}^{s} \binom{s}{i} {p_c}^i (1-p_c)^{s-i}$. The probability P_c that f nodes collude increases whenever the data owner generates a new shard and allocates it to a new node. On the contrary, P_c never decreases. Indeed, we cannot assume that nodes leaving the system will not re-join in the future and that they do not have a copy of the shard initially allocated to them, even in case the data owner closed the contract. A malicious node can keep a copy of the data on purpose, to prevent resource deletion and possibly sell the shard to non-authorized users.

The owner can reduce P_c only by re-encrypting r with a new key. This process is however quite expensive as it requires to locally reconstruct r (downloading f shards), decrypt it, re-encrypt its plaintext content with a new encryption key, encode the resulting ciphertext, and distribute the new set of s shards to s nodes. To limit such overhead, our solution is based on the definition of a threshold P_c^{max} for P_c, representing the maximum probability that the owner can tolerate that r is exposed. When, as a consequence of the generation of a shard, P_c exceeds P_c^{max}, the owner re-encrypts r. Clearly, shards generated before resource re-encryption cannot be used together with shards generated after re-encryption for resource reconstruction, hence nullifying possible misbehaviors by nodes whose contract has been closed before re-encryption.

The bottom chart in Figure 4 illustrates the evolution of P_c due to corrective actions taken by the owner for keeping P_u below P_u^{max} (i.e., the generation of new shards). As long as P_c remains below threshold P_c^{max}, no corrective action is taken (green circles 1 and 2 in the figure). In the example we set P_c^{max} to tolerate the generation of two shards. When the generation of a new shard causes P_c to reach P_c^{max} (red triangle 5 and green circle 4 in the figure), the owner re-encrypts r. Resource re-encryption resets both P_u and P_c to their initial value (green circle 5 in the figure). The lower is P_u^{max}, the more frequently shards will be generated and the more frequently the data owner will need to re-encrypt r.

V. Implementation and Experiments

To assess the benefit of using fountain codes in contrast to Reed-Solomon used by real world DCS networks, we implemented our proposal on top of Storj [17]. Among the existing DCS, we selected Storj because it is one of the leaders in the DCS scenario, and its open-source implementation easily permits the integration of our solution. Storj relies on Reed-Solomon to guarantee resource availability, and applies encoding and decoding at the owner and user side, respectively. We then modified the Storj client to adopt fountain codes instead of Reed-Solomon. Specifically, we used RaptorQ code (specified in IETF RFC 6330 [12]), which is the latest evolution of Rapid Tornado (Raptor) codes [16]. RaptorQ is among the most effective fountain codes available to date, since the probability of not being able to reconstruct a resource when accessing $f+\varepsilon$ shards is negligible, $0.01^{\varepsilon+1}$ (i.e., 1% accessing f shards, 0.01% accessing $f+1$ shards).

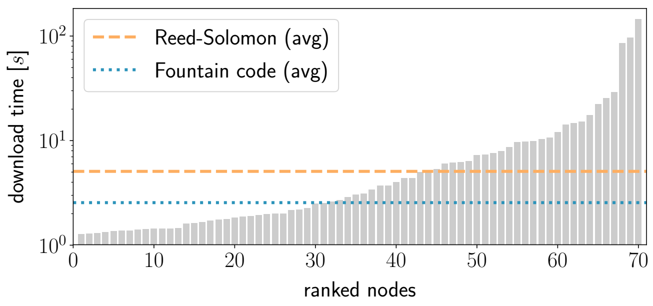


Fig. 5: Download time of a 10MB shard from each node that has been contacted in the test (in increasing order of time)

To analyze the performance of our solution, we set parameters f=7 and s=10 and considered shards of size 100kB, 1MB, and 10MB (i.e., resources of size 700kB, 7MB, and 70MB, respectively). In our experiments, we considered p_u=0.015, $P_u^{min}=10^{-12}$, $P_u^{max}=10^{-5}$, p_c=0.025, and $P_c^{max}=10^{-6}$. These parameter values support the generation of 22 shards before requiring resource re-encryption. The client used for our experimental evaluation is an Intel i7 4770K with 16 GB RAM, running Ubuntu 18.04, connected to a network with enough bandwidth to download all the required (f=7) shards in parallel. In our tests, we contacted 70 nodes. The histogram bars in Figure 5 illustrate the time necessary to download a shard of 10MB from each of these nodes. The nodes are reported on the x-axis in increasing download time. In the experiments, we focused on download time as the use of AONT causes a negligible overhead, since the bottleneck is represented by the network (e.g., the solution in [2] has 2.5 GB/s throughput, which is orders of magnitude higher than fast network connections).

If the user interested in downloading the resource knows exactly the response time of each node in the DCS, she can choose the optimal pool of nodes, and minimize the time for retrieving the shards necessary to reconstruct r. When using RaptorQ, the best strategy consists in contacting the f fastest nodes, since any subset of f shards enables resource reconstruction. Hence, the download time is given by the access time of the f-th node in the ranking of the s nodes storing a shard of the resource. When using Reed-Solomon the best strategy consists in contacting the f fastest nodes that store f different shards, which might not be the first f nodes in the ranking in case of duplicated shards. The download time then is given by the access time of the slowest contacted node.

Since usually clients do not know the response times of the nodes in the DCS, the best strategy consists in contacting all the s nodes in parallel, and stop ongoing downloads when the downloaded shards enable resource reconstruction (i.e., after the first f shards have been downloaded for RaptorQ,

Shard size	Reed-Solomon	RaptorQ
100 kB	0.680s ($\sigma = 0.220$)	0.487s ($\sigma = 0.167$)
1 MB	1.071s ($\sigma = 0.306$)	0.758s ($\sigma = 0.234$)
10 MB	5.082s ($\sigma = 3.232$)	2.552s ($\sigma = 0.962$)

TABLE I: Average time required to download enough shards from the network to reconstruct a resource

after f different shards have been downloaded with Reed-Solomon). Table I reports the average over 100 runs of the time (and standard deviation σ) for downloading the shards necessary to reconstruct a resource obtained when using Reed-Solomon and RaptorQ. As expected, RaptorQ provides better response times, with a reduction between 28.32% (with shards of 100kB) and 49.78% (with shards of 10MB), than Reed-Solomon. This difference is due to the fact that the download time is a function of both the download throughput and the latency of nodes. For smaller shards, latency is more relevant than for larger shards, where node bandwidth represents the factor contributing most to download times.

Figure 5 reports the average time necessary to reconstruct the resource in case of shards of 10MB when using RaptorQ (blue line) and Reed-Solomon (yellow line). As illustrated in the figure, RaptorQ requires to acquire shards from $\sim$45-th percentile of the nodes (30 out of 70 nodes), while for Reed-Solomon requires to acquire shards from $\sim$65-th percentile (45 out of 70 nodes), causing a delay in resource reconstruction. We conclude that the adoption of fountain codes provides a general advantage, thanks to the possibility for the final user to rely on faster nodes for resource reconstruction, while providing security guarantees.

VI. Related work

The problem of providing availability guarantees to data stored in DCS networks has been considered by real-world systems. Storj [17], Sia [18], SAFE Network [7] adopt techniques based on Reed-Solomon, and Enigma [1] relies on fountain codes. Although these DCS networks also pay attention to security, they do not aim at protecting data against coalitions of malicious nodes. Moreover, the network immediately reacts to any configuration change. Our proposal is orthogonal to these solutions, and can be easily integrated in real world DCS networks (as illustrated in Section V) to balance availability and security guarantees while limiting the data owner's intervention.

The problem of balancing availability and security in DCS networks has been recently addressed in [3]. This proposal is based on the combined adoption of AONT and data replication, and introduces two allocation strategies of shards to nodes. The analysis of the proposed allocation strategies illustrates how they can be tuned to balance availability and security. Unlike our approach, this proposal considers a static scenario and adopts replication.

The combined adoption of AONT and error-correcting code techniques has been recently explored to the aim of protecting outsourced data against possibly curious storage providers and offering high performance (e.g., [4], [14]). *AONT-RS* [14] combines Rivest's AONT [15] with Reed-Solomon, while *AONT-LT* [4] uses Luby Transform code (which is a class of fountain codes [6], [16]) instead of Reed-Solomon. Although these proposals present similarity with our approach, they are specifically focused on static scenarios, where the set of nodes is fixed and nodes are not expected to frequently leave/join the network. These solutions are then suited to cloud-of-clouds but not to DCS scenarios.

VII. Conclusions

We presented an approach aimed at balancing availability and security of resources stored in a DCS, where nodes naturally leave and re-join the system. The proposed solution combines AONT encryption and fountain codes. Our approach is based on the definition of thresholds for both availability and security guarantees, and on the idea that the data owner intervenes only when the thresholds are violated. Our experimental evaluation demonstrates that our proposal considerably reduces the time required to reconstruct resources in a DCS, while limiting the exposure to coalitions of malicious nodes.

References

[1] Anglano, C., Gaeta, R., Grangetto, M.: Exploiting rateless codes in cloud storage systems. IEEE TPDS **26**(5), 1313–1322 (2015)

[2] Bacis, E., De Capitani di Vimercati, S., Foresti, S., Paraboschi, S., Rosa, M., Samarati, P.: Mix&Slice: Efficient access revocation in the cloud. In: Proc. of CCS. Vienna, Austria (October 2016)

[3] Bacis, E., De Capitani di Vimercati, S., Foresti, S., Paraboschi, S., Rosa, M., Samarati, P.: Securing resources in decentralized cloud storage. IEEE TIFS (2019, pre-print)

[4] Baldi, M., Maturo, N., Montali, E., Chiaraluce, F.: AONT-LT: A data protection scheme for cloud and cooperative storage systems. In: Proc. of HPCS. Bologna, Italy (July 2014)

[5] Benet, J.: IPFS - content addressed, versioned, P2P file system. Tech. rep., Protocol Labs (2014)

[6] Cataldi, P., Shatarski, M.P., Grangetto, M., Magli, E.: Implementation and performance evaluation of LT and Raptor codes for multimedia applications. In: Proc. of IEEE MSP. Pasadena, CA, USA (December 2006)

[7] Irvine, D.: Maidsafe distributed file system. Tech. rep., MaidSafe (2010)

[8] Kapusta, K., Memmi, G.: Circular AON: A very fast scheme to protect encrypted data against key exposure. In: Proc. of CCS. Toronto, Canada (October 2018)

[9] Karame, G.O., Soriente, C., Lichota, K., Capkun, S.: Securing cloud data under key exposure. IEEE TCC pp. 1–13 (2017, pre-print)

[10] Labs, P.: Filecoin: A decentralized storage network. Tech. rep., Protocol Labs (2017), http://filecoin.io/filecoin.pdf

[11] Lambert, N., Bollen, B.: The SAFE network - a new, decentralised internet. Tech. rep., MaidSafe (2014)

[12] Luby, M., Shokrollahi, A., Watson, M., Stockhammer, T., Minder, L.: RaptorQ forward error correction scheme for object delivery – RFC 6330. IETF Request For Comments (2011)

[13] Reed, I., Solomon, G.: Polynomial codes over certain finite fields. Journal of the Society for Industrial and Applied Mathematics **8**(2), 300–304 (June 1960)

[14] Resch, J.K., Plank, J.S.: AONT-RS: Blending security and performance in dispersed storage systems. In: Proc of FAST. San Jose, CA, USA (February 2011)

[15] Rivest, R.L.: All-or-nothing encryption and the package transform. In: Proc. of FSE. Haifa, Israel (January 1997)

[16] Shokrollahi, A.: Raptor codes. IEEE/ACM TON **6**(3-4), 213–322 (May 2011)

[17] Storj Labs Inc.: Storj: A decentralized cloud storage network framework. Tech. rep., Storj Labs Inc. (2018)

[18] Vorick, D., Champine, L.: Sia: Simple decentralized storage. Tech. rep., Nebulous Inc. (2014)