Supporting Data Owner Control in IPFS Networks

Marco Abbadini*, Michele Beretta*, Sabrina De Capitani di Vimercati†, Dario Facchinetti*,
Sara Foresti†, Gianluca Oldani*, Stefano Paraboschi*, Matthew Rossi*, Pierangela Samarati†
* Università degli Studi di Bergamo, Italy – Email: *firstname.lastname*@unibg.it
† Università degli Studi di Milano, Italy – Email: *firstname.lastname*@unimi.it

***Abstract*—Decentralized storage architectures are emerging as valid complementary solutions to cloud-based storage services. InterPlanetary File System (IPFS) is one of the most well-known distributed file storage protocols with wide adoption, good performance, and a variety of applications built over it. However, IPFS does not natively support data confidentiality and its decentralized nature limits the ability of data owners to maintain control on their resources and to force their deletion.**

We propose Mix-IPFS, an approach that allows data owners to maintain control on their resources uploaded to IPFS, guaranteeing their confidentiality and supporting secure deletion. Mix-IPFS is based on AONT encryption, which has the nice property of preventing decryption if the whole ciphertext is not available. Data owners can permanently delete a resource by making a small portion of its encrypted representation unavailable. Our solution uses a virtual file system to guarantee transparency to data owners (i.e., they can operate on plaintext resources). The experimental evaluation shows that the overhead of our approach is negligible (less than 2% for both upload and access operations).

***Index Terms*—InterPlanetary File System, Encryption, AONT**

I. Introduction

Since their origins, Information and Communication Technologies have been characterized by a spectrum of options, with systems controlled by a single or a few powerful companies on one side, and open architectures that support the activities of a multitude of actors on the other side. In several domains, the evolution of technology has transformed scenarios that saw a single or dominant actor into a landscape where multiple companies and individuals can contribute. As technology evolves, components become more standardized, and the cost required to offer services decreases, an opportunity arises to build open services that replace what traditionally was offered by a relatively small number of companies. Networks, with their increasing efficiency and pervasiveness, have played a significant role in this evolution.

A domain where this evolution may occur is cloud storage. The role of cloud storage providers is significant and several services are offered today by major players in the IT world (e.g., Amazon S3 and Dropbox). Cloud storage services are well-engineered and highly performant, even if they sometimes exhibit critical failures [1]–[3]; the design of novel protocols can extend the opportunities available to users and facilitate the success of cloud storage solutions based on the participation of a large number of entities.

Decentralized storage architectures are emerging as a robust and scalable solution for storing resources. A solution that is particularly interesting is *InterPlanetary File System* (IPFS), a protocol that supports the dissemination of resources according to the P2P paradigm and represents the largest and most widely used *Decentralized Web* platform. In general, there is a continuous interest in the design and development of IPFS, as shown by the wide deployment and variety of applications built over it [4]. The IPFS infrastructure has been deployed in over 2700 Autonomous Systems, across 464k IP addresses, covering 152 countries. IPFS is seeing widespread uptake with more than 3M web client accesses and beyond 300k unique nodes serving content in the network every week. Although content retrieval in IPFS is slower than direct HTTP access, delays are still reasonable for a number of use cases. For instance, 75% of retrievals from Europe are under 2 s (including looking up the content host and fetching a 0.5 MB file). Moreover, the use of gateway caching can substantially reduce retrieval latency, with 76% of the requests being served in less than 250 ms.

A crucial aspect in the realization of any storage service, especially when it offers network access to resources, is the ability to protect resource confidentiality. By design, IPFS focuses on reliability and integrity, but does not provide confidentiality of resources, which should be protected by the owner before storing them on IPFS. Also, being distributed and replicated, IPFS does not guarantee permanent resource deletion, and hence owners lose control on their resources.

In this paper, we present an approach, called *Mix-IPFS*, that protects the confidentiality of resources and allows their owners to maintain the control on the resources uploaded to IPFS. Our proposal applies an *All-Or-Nothing-Transform* (AONT) encryption mode, which ensures that the whole encrypted resource is needed to reconstruct the corresponding plaintext. Our solution is transparent and fully integrated within the file system. We implement a virtual file system that automatically: splits the resource in IPFS chunks; encrypts each chunk; and uploads the encrypted resource chunks to IPFS. Our file system locally stores a small portion (fragment) of each encrypted resource, which is not stored in IPFS, to enable permanent deletion. When the data owner wants to delete a resource, the file system deletes the locally stored fragment, making the reconstruction of the plaintext resource impossible from the chunks stored in IPFS. Our solution offers the opportunity to reduce economic costs using spare storage space made available by third parties and, potentially, ease the permanent-storage requirement. Our experiments show that the overhead of Mix-IPFS on access times is limited (less than 2%) while not affecting performance.

Mix-IPFS's source code is available at https://github.com/unibg-seclab/ipfs-owner-control.

II. Basics concepts

The two building blocks of our solution are the adoption of IPFS for data storage and of AONT for data encryption.

IPFS. IPFS is an open-source distributed data storage network that relies on a content-based P2P network [4], [5]. Each IPFS node is identified by a *PeerID*. A resource stored in IPFS is split in chunks of fixed size (by default 256 KiB), each identified by a unique *Content Identifier* (CID). The CID of a chunk contains the result of a cryptographic hash function applied on the chunk, and other metadata. The chunks composing a resource are organized in a Merkle Directed Acyclic Graph (Merkle DAG), where the CID of each node is obtained by hashing the CIDs of its children. The CID of the root of the Merkle DAG is the resource CID. The use of a cryptographic hash function and of the Merkle DAG structure provides data immutability and self-certification. In fact, any IPFS user can autonomously verify whether a resource has been modified by checking the resource CID against the resource content (i.e., a resource cannot be modified without changing its CID). A Distributed Hash Table (DHT) maintains the association between a CID and the PeerID(s) where the corresponding resource is stored. To publish a new resource, the IPFS node of the owner splits the resource in chunks, builds the Merkle DAG, and pushes a new record for the resource CID in the DHT, in association with its PeerID. To retrieve a resource identified by a CID, an IPFS node retrieves from the DHT the PeerIDs of the IPFS nodes providing the CID, fetches the corresponding chunks, and verifies whether the resource matches its CID. Once a resource has been accessed, the IPFS node can make it accessible as a replica. While the publication of a resource involves the P2P network, the deletion of a resource is a local operation: if the resource has been replicated it remains available.

AONT. AONT is an encryption mode that transforms a plaintext resource into an encrypted resource guaranteeing complete interdependence among all the bits of the encrypted resource (i.e., each bit of the output depends on every bit of the input). Complete mixing implies that missing even a small portion of the ciphertext prevents reconstruction of the plaintext resource, even if the encryption key is known. Among the different proposals for implementing an AONT (e.g., [6]–[10]) we leverage the Mixing structure described in [8], which can benefit from the hardware accelerated implementation of AES available in most modern CPUs and ensures protection also in scenarios where users from which access should be prevented know the encryption key.

III. Mix-IPFS

While using IPFS for resource storage and management provides a number of advantages (e.g., data availability, limited economic costs), its wide adoption might be restricted by the loss of control by the owner, who cannot permanently delete resources. To support resource deletion while maintaining the advantages of IPFS, Mix-IPFS encrypts a resource with AONT. Intuitively, using AONT, the resource owner can force resource deletion by deleting a portion of the encrypted resource. To maintain full control, this portion, which we call *golden fragment*, is locally stored only at the owner side (i.e., not outsourced). Protection of cryptographic secrets is out of the scope of our work, however existing works address this problem (e.g., [11]). We now describe how Mix-IPFS uploads and access resources in IPFS.

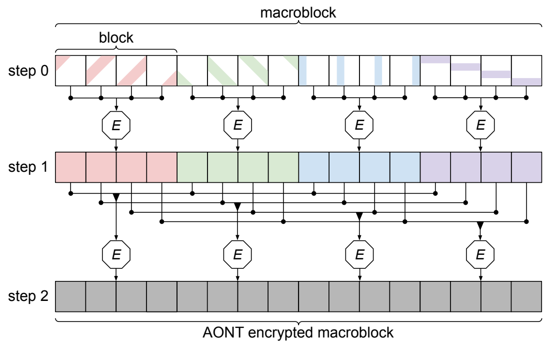


Fig. 1: AONT mixing scheme

Upload. A resource that needs to be stored in the IPFS network is first split into a set $\{M_1, \ldots, M_m\}$ of *macroblocks*, whose size must be a power-of-2 of the size of the block input to the symmetric block cipher at the basis of the working of Mixing. Mixing encrypts each macroblock through a sequence of symmetric encryption steps. At each step, the block input to the block cipher is obtained combining bits from different blocks resulting from the previous step. This guarantees, after a well defined number of steps, complete interdependence among the bits in the encrypted macroblock. Figure 1 illustrates an example of encryption of a macroblock composed of $4 = 2^2$ blocks, using a block cipher E (e.g., AES) that takes as input a block of size 16 bytes. In the first step, each block of the plaintext macroblock is separately encrypted. In the second step, the four blocks input to the four encryption operations E are obtained combining a different quarter (e.g., the first 4 bytes of each block for the first E) of each of the four output blocks of the previous step. In this way, each bit in the four output blocks depends on each bit in the four input plaintext blocks, as visible from the color-coding in the figure. The number p of encryption steps necessary for mixing a macroblock depends on the size of the macroblock, the size b of the block, the number n of blocks mixed at each step (4 in our example). In particular, p steps guarantee the mixing of a macroblock of size $n^p \cdot \frac{b}{n}$ bytes. For instance, a macroblock of size 256 KiB requires 8 steps of encryption (i.e., 256 KiB $= 4^8 \cdot 16/4$ bytes).

For each macroblock M_i, $i = 1, \ldots, m$, the owner locally stores a small portion of arbitrary size of its encrypted representation. The collection of all these portions of the macroblocks forming a resource represents the golden fragment of

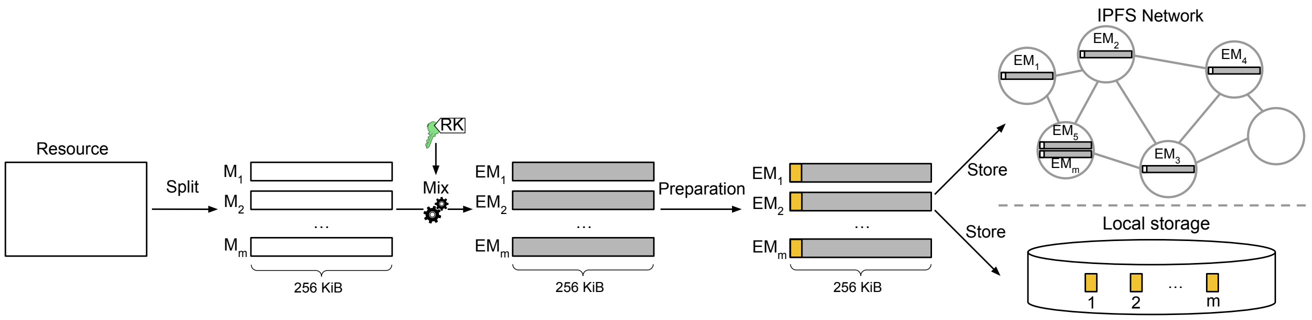


Fig. 2: Resource upload with Mix-IPFS

the resource. For each encrypted macroblock, the bits locally stored are replaced in the macroblock with all 0s. While in principle the owner can locally store any sequence of bits from a macroblock, for simplicity, in our implementation we store in the golden fragment the initial sequence of bits of each macroblock. The resulting macroblocks are then uploaded to the IPFS network. Note that the size of macroblocks and IPFS chunks can be independently chosen by the resource owner. However, it is convenient, in terms of access time, to use macroblocks of the same size as IPFS chunks. Figure 2 illustrates the process of publishing a resource in an IPFS network using our approach, assuming macroblocks of 256 KiB and using key RK for encryption.

Access. To access a resource, the process illustrated in Figure 2 is reverted. A user first retrieves the chunks composing the resource identified by a given CID, and gets the locally stored golden fragment. When the download of a chunk has been completed, the user can replace the initial sequence of 0s with the corresponding bits in the golden fragment. The macroblock can then be decrypted (applying Mixing in decryption mode). The overhead of Mixing is negligible compared to the application of a simple direct AES encryption on the IPFS chunks (see Section V). Also the size of the golden fragment is expected to be negligible compared to the size of the chunks.

IV. Mix-IPFS Architecture

We now illustrate the architectural design and working of Mix-IPFS. In the design of Mix-IPFS, we aim at a system that allows for transparency, that is, hiding to owners the adoption of AONT encryption for protecting resources and of IPFS for their storage. In the following, we will use the term *user* to generically refer to the user of Mix-IPFS, which is the owner uploading resources, and the user accessing them.

A. Architectural design

Our solution relies on a virtual file system that represents the interface through which a user interacts with the IPFS network to upload and access resources. Transparency is guaranteed by providing the user with a directory in the file system, listing resources stored in IPFS using our protection technique. Mix-IPFS is then responsible for mediating store and access requests (i.e., to transparently apply encryption/decryption operations and to manage the golden fragment).

Mix-IPFS uses FUSE (*Filesystem in USErspace*) [12] as a virtual file system, since it is a well-known Linux kernel technology that allows unprivileged users to access virtual file systems without the need to modify the kernel code or load new modules. Also, FUSE has the advantage of being compatible with all Unix and Unix-like systems (e.g., FreeBSD, macOS) as well as with Windows [13]. FUSE consists of two modules: *i)* the FUSE Linux kernel module, and *ii)* the `libfuse` library [14], which operates in userspace and provides a reference set of APIs used by the Linux kernel module to serve file system-related requests. Mix-IPFS provides the APIs defined by `libfuse` necessary to handle upload and access operations, work with symbolic links, and modify resource metadata (i.e., file attributes including, for example, access privileges, last accessed and modified timestamps). Mix-IPFS has been designed with performance in mind, aiming at minimizing access times. To this purpose, Mix-IPFS organizes information about resources in a trie data structure, which enhances performance of prefix-based searches in the directory.

Before mounting the file system, Mix-IPFS authenticates the user through the user master password (MP), which is also used to derive a master key (MK) that is adopted for encrypting Mix-IPFS metadata (Figure 3, step A). Mix-IPFS metadata include the relevant structures needed for the functioning of the virtual file system, such as attributes and names of files, cryptographic keys used for resource encryption/decryption (i.e., RK), and identifiers of the data stored in IPFS (i.e., CIDs of resources and reference to the golden fragments). If authentication succeeds, Mix-IPFS loads (or initializes if it is not already created) the file system metadata and, based on them, mounts the virtual file system in a dedicated directory (Figure 3, step B). The user can then interact with the directory in the same way as with any other directory in the file system. User interactions are however captured by the Linux kernel and redirected to the FUSE module via the VFS (*Virtual Filesystem Switch*) module. The FUSE module serves the user's requests through the `libfuse` library.

B. Management of upload and access operations

We now illustrate how the insertion of a resource into the local directory of the user file system and the access to a resource are managed by Mix-IPFS. Figures 3 and 4 show the

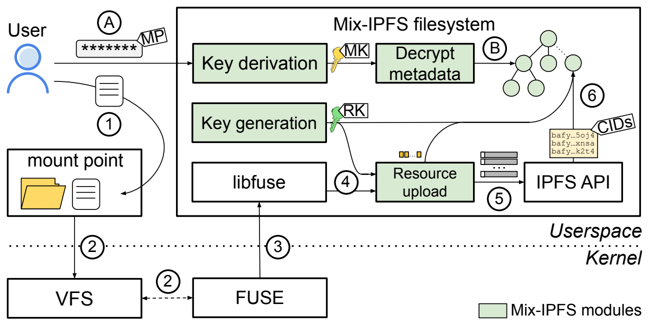


Fig. 3: Mount Mix-IPFS and upload a resource

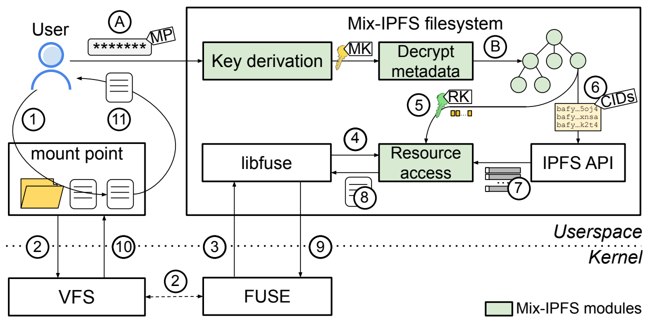


Fig. 4: Access a resource in Mix-IPFS

working of the upload and access operations. The green (gray in black and white printout) modules in the figures are specific of Mix-IPFS.

Upload. When a user inserts a new resource into the local directory (step 1 in Figure 3), VFS intercepts the request and passes it to FUSE (step 2). The request is then forwarded to Mix-IPFS (step 3). Mix-IPFS first encrypts the resource as described in Section III, keeping the macroblock size the same as IPFS chunks (256 KiB each) and keeping as golden fragment the first 1024 bytes of each macroblock (step 4). Mix-IPFS then asynchronously loads the encrypted macroblocks to IPFS nodes and annotates the CIDs (calculated by IPFS) of the encrypted macroblocks (step 5). Mix-IPFS finally maps the name of the resource to a tuple of the form ⟨CID, `out_loc`⟩, where CID is the CID of the resource and `out_loc` is a token (e.g., a path on the local file system) necessary to retrieve the golden fragment, and updates metadata (step 6). Note that the information locally stored at the user-side to support access to resources includes the golden fragment, the encryption key RK used by Mixing, tuple ⟨CID, `out_loc`⟩, and the size of the portion extracted from macroblocks to build the golden fragment.

Access. When the user wants to access their resource (step 1 in Figure 4), VFS intercepts the request and passes it to FUSE (step 2). The request is then forwarded to Mix-IPFS (step 3), which reverses the protection applied when the resource has been uploaded to IPFS. Mix-IPFS then first retrieves the (locally stored) tuple ⟨CID, `out_loc`⟩ for the resource of interest (steps 4 and 5) and downloads, from the IPFS network, the resource with the identified CID (steps 6 and 7). Mix-IPFS then operates in parallel on all the retrieved macroblocks: it replaces the initial sequence of 0s with the corresponding sequence of bits in the golden fragment and applies AONT Mixing in decryption mode (step 8). The plaintext resource is then returned to the user (steps 9–11).

Mix-IPFS easily manages also updates to resources by combining the upload and access processes illustrated above. In fact, any update to an IPFS chunk implies an update to its CID, and hence to the CID of the resource. Delete operations, as already noted, are managed by permanently removing the golden fragment(s) of the resource.

V. Experimental Evaluation

To assess the performance overhead introduced by Mix-IPFS, we conducted a wide experimental analysis comparing access times to files stored in IPFS in plaintext, when using a traditional block cipher, and when using Mix-IPFS (Section V-A). We also analyzed the local storage overhead due to the golden fragment and additional file system metadata (Section V-B).

For the experimental evaluation, we used a machine with Ubuntu 22.04 (kernel 5.19), IPFS 0.19.1, `libfuse` 3.10.5, and equipped with an AMD Ryzen 9 7900X, 64 GB of RAM, and 2 TB of SSD. The machine is located in southern Europe (note that IPFS performance varies depending on the region, as shown in [4]). The resources used for our experiments have been replicated on different public IPFS nodes.[1] In our experimental evaluation, we used 10 different IPFS nodes to test the benefit of the parallelization of our Mixing, which leverages the distributed nature of IPFS. We considered IPFS chunks of 256 KiB and assumed to locally store in the golden fragment 1 KiB for each macroblock.

A. Latency

To analyze the latency of Mix-IPFS, which represents a crucial aspect when accessing resources, we consider two scenarios that differ in how resources are protected: *1)* encryption through AES in CBC and CTR mode; and *2)* encryption through Mixing. Note that, in our experiments, AES operates at the macroblock level, to leverage the same parallelism implemented in our solution and enable a fair comparison among the considered scenarios. We analyzed separately the overhead introduced by the protection techniques of these two scenarios when uploading and when accessing a resource. The overhead is computed with respect to the case where resources are stored in IPFS in plaintext, which is our baseline. The results illustrated in the following have been obtained as the average of 20 runs. The colored areas in the figures represents the standard deviation obtained in the experimental evaluation.

Upload. The upload of a resource can be considered as operating in two independent steps: *i)* protect the resource, and *ii)* publish the protected resource on IPFS. Since IPFS

[1]List of public IPFS nodes: https://ipfs.github.io/public-gateway-checker/

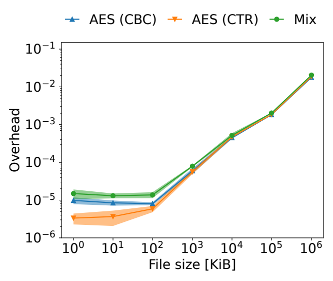
natively implements the second step, and the size of the encrypted resource does not change significantly, the overhead introduced when uploading a resource only depends on the performance of the protection technique adopted (i.e., AES in CBC and CTR mode or Mixing). Considering a resource of size between 1 KiB and 10^6 KiB, Figure 5a illustrates the average overhead, computed as the ratio between the time necessary to protect the resource and our baseline (i.e., the time spent to publish the resource on IPFS), when using AES in CBC and CTR mode and when using Mixing. As expected, the overhead increases with the size of the resource, since the time spent for encrypting a resource depends on its size, while IPFS publication is less susceptible to it. We note, however, that the overhead of Mixing is similar to the overhead of traditional AES encryption (2% overhead for 1GB resources), while providing higher protection guarantees.

Access. Similarly to the upload operation, the access to a resource operates in two steps: *i)* retrieve and download from IPFS the chunks composing the resource, which is natively implemented in IPFS, and *ii)* remove the protection layer. Considering a resource of size between 1 KiB and 10^6 KiB, Figure 5b illustrates the average overhead, computed as the ratio between the time necessary to remove the protection layer (i.e., AES in CBC and CTR mode or Mixing), and the time spent to retrieve the resource (baseline). The overhead increases with the size of the resource, but the increase is less steep compared to what observed for upload operations in Figure 5a. This smoother trend is due to the fact that IPFS takes longer to fetch large resources (step *i)* above), as shown in Figure 6b that reports the average absolute time necessary to download a resource varying its size. Figure 5b confirms that our approach is competitive with AES also for access operations. For small resources, which cannot take advantage of parallelization since they fit in one macroblock, we observe a higher cost when using Mixing compared to AES. The performance overhead is however limited (0.95% compared to AES in CBC mode, 0.85% compared to AES in CTR mode). This is confirmed by Figure 6a, showing the average absolute time for removing the protection layer (step *ii)* above).

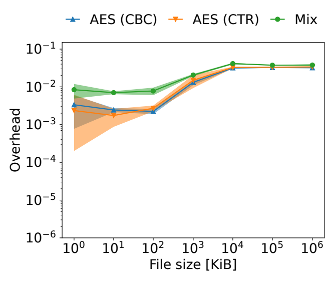
B. Storage

We also analyze the additional space needed for the adoption of Mix-IPFS (i.e., the space required for storing the golden fragment and the file system metadata).

Golden fragment. Assuming to locally store 1 KiB for each macroblock and to use IPFS chunks (and macroblocks) of 256 KiB, Figure 7a illustrates the overhead, computed as the ratio between the size of the golden fragment and the size of the (plaintext) resource, varying the size of the resource between 1 KiB and 10^6 KiB. As expected, the overhead due to the local storage of the golden fragment is higher for small resources, especially for resources having size smaller than 256 KiB. In fact, Mix-IPFS applies padding to reach a size that is a multiple of the macroblock size before encrypting the resource. For instance, considering a resource of size 20

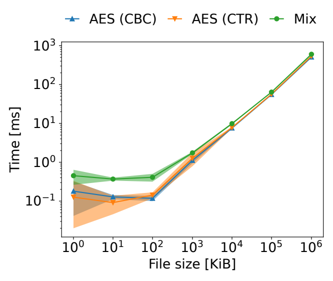


(a) Upload overhead

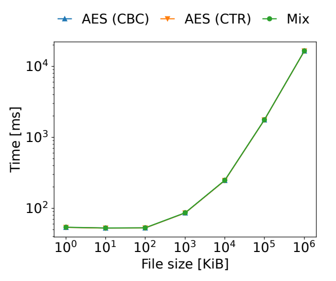


(b) Access overhead

Fig. 5: Average overhead of Mixing and of AES in uploading and accessing a resource, varying the size of the resource

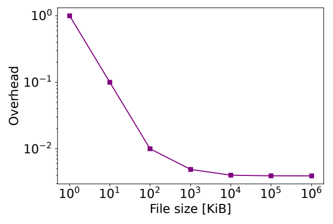


(a) Decryption time

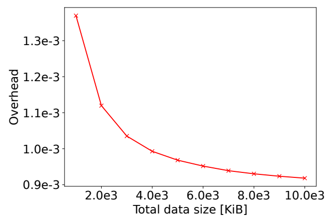


(b) Download time

Fig. 6: Average absolute times for accessing a resource, varying the size of the resource



(a) Golden fragment overhead



(b) Metadata overhead

Fig. 7: Space overhead of the golden fragment and of the file system metadata

KiB, the overhead of the golden fragment is $1/20$. The figure shows that the storage overhead quickly approaches constant value $1/256$ (i.e., 0.39% of the resource size).

Metadata. Figure 7b illustrates the overhead, computed as the ratio between the on-disk size of file system metadata and the size of the plaintext resources handled by Mix-IPFS, varying the overall size of plaintext resources between 10^3 KiB and 10^4 KiB. The overhead remains extremely low when managing small sets of resources (less than 0.14% in our experiments), and decreases as the overall size of resources grows.

VI. Related work

The problem of protecting data confidentiality in modern P2P networks and decentralized storage networks has been recently addressed by several approaches (e.g., Freenet [15], Filecoin [16], Arweave [17], and Sia [18]). While presenting similarities with Mix-IPFS, these approaches do not allow the resource owners to be in control of resource deletion. The work in [19] addresses the problem of scheduling the release of confidential data in a decentralized network, even when some of the nodes may not follow the P2P protocol. Contrary to our scenario, this solution relies on economic incentives and penalties to guarantee persistent storage to the client.

The use of AONT for providing data confidentiality in decentralized systems has been recently addressed, with approaches that enforce client-side encryption in AONT mode. The proposals in [20], [21] introduce a model based on AONT and data replication for providing both security and availability guarantees to resources stored in a decentralized cloud storage. SAFE network [22], [23] is a decentralized cloud storage network that adopts convergent encryption for data protection. These solutions, while effective in providing data confidentiality, differ from our approach since they rely on the nodes in the network for enforcing resource deletion, which is therefore not fully under the owner control. Similarly, the adoption of linear network coding aims at protecting resources stored in a distributed cloud storage by splitting each resource into multiples fragments allocated at different nodes [24]. The adoption of a scheme similar to secret sharing for splitting resources provides data confidentiality, but resource deletion is not under the owner control.

The problem of providing a file system that transparently manages security mechanisms has been widely studied (e.g., [25], [26]). Among existing solutions, EncFS [25] offers an encrypted userspace file system relying on FUSE. OramFS [26] is instead an encrypted (and optionally authenticated) Oblivious RAM file system. Differently from our proposal, these approaches do not adopt AONT encryption mode to guarantee strong interdependency, and are not designed to operate on P2P networks.

Another line of works related to our proposal is aimed to provide confidentiality of outsourced data (e.g., [27], [28]). The approach in [29] provides protection of confidential information by maintaining a portion of the data at the owner side. This solution completely departs from encryption, assuming that what is sensitive is the association among data.

VII. Conclusions

We proposed Mix-IPFS, an approach that aims at protecting confidentiality of data stored in IPFS, and at enabling owners to remain in control of their resources and to force delete operations over them. Data confidentiality is guaranteed by applying AONT encryption at the owner-side, and the control over delete operations is preserved by locally storing a small fragment of the encrypted resource. Being integrated in the file system, Mix-IPFS guarantees transparency to users. Our experimental evaluation confirms the limited overhead of Mix-IPFS, both in terms of access time and of local storage space.

Acknowledgment

This work was supported in part by the EC under projects GLACIATION (101070141) and MARSAL (101017171), by the Italian MUR under PRIN project POLAR, and by projects GRINS (PE00000018) and SERICS (PE00000014) under the NRRP MUR program funded by the EU – NGEU.

References

[1] M. Rosemain and R. Satter, "Millions of websites offline after fire at French cloud services firm," https://www.reuters.com/article/us-france-ovh-fire-idUSKBN2B20NU, 2021.

[2] D. Coldewey, "Cloudflare DNS goes down, taking a large piece of the internet with it," https://techcrunch.com/2020/07/17/cloudflare-dns-goes-down-taking-a-large-piece-of-the-internet-with-it, 2020.

[3] E. Mathews, "Amazon cloud outage hits major websites, streaming apps," https://www.reuters.com/article/amazon-com-outages-idCAKBN2IM1U0, 2021.

[4] D. Trautwein, A. Raman, G. Tyson, I. Castro, W. Scott, M. Schubotz, B. Gipp, and Y. Psaras, "Design and evaluation of IPFS: A storage layer for the decentralized web," in *Proc. of SIGCOMM*, August 2022.

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

[6] R. L. Rivest, "All-or-nothing encryption and the package transform," in *Proc. of FSE*, January 1997.

[7] M. Bellare and P. Rogaway, "Optimal asymmetric encryption," in *Proc of EUROCRYPT*, May 1994.

[8] 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*, October 2016.

[9] ——, "Mix&Slice for efficient access revocation on outsourced data," *IEEE 2TDSC*, 2023, *Early Access*.

[10] V. Boyko, "On the security properties of OAEP as an All-or-Nothing transform," in *Proc. of CRYPTO*, 1999.

[11] M. Abbadini, M. Beretta, D. Facchinetti, G. Oldani, M. Rossi, and S. Paraboschi, "Lightweight cloud application sandboxing," in *Proc. of CLOUDCOM*, 2023.

[12] FUSE Project, "FUSE: The Linux kernel documentation," https://www.kernel.org/doc/html/next/filesystems/fuse.html, 2023.

[13] WinFsp, "Windows file system proxy," https://winfsp.dev/, 2023.

[14] FUSE Project, "libfuse: Reference implementation for communicating with the fuse kernel module," https://github.com/libfuse/libfuse, 2023.

[15] The Freenet Project, "Freenet," https://freenetproject.org/, 2023.

[16] Filecoin, "Filecoin," https://filecoin.io/, 2023.

[17] Arweave, "Meet Arweave: Permanent information storage," https://www.arweave.org/, 2023.

[18] Sia, "Sia, decentralized data storage," https://sia.tech/, 2023.

[19] E. Bacis, D. Facchinetti, M. Guarnieri, M. Rosa, M. Rossi, and S. Paraboschi, "I told you tomorrow: Practical time-locked secrets using smart contracts," in *Proc. of ARES*, August 2021.

[20] E. Bacis, S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, M. Rosa, and P. Samarati, "Securing resources in decentralized cloud storage," *IEEE TIFS*, vol. 15, no. 1, December 2020.

[21] ——, "Dynamic allocation for resource protection in decentralized cloud storage," in *Proc. of GLOBECOM*, December 2019.

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

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

[24] M. Sipos, F. H. Fitzek, D. E. Lucani, and M. V. Pedersen, "Distributed cloud storage using network coding," in *IEEE CCNC*, January 2014.

[25] V. Gough, "EncFS: an Encrypted Filesystem for FUSE," https://vgough.github.io/encfs/, 2023.

[26] Kudelski Security, "OramFS: ORAM filesystem written in Rust," https://github.com/kudelskisecurity/oramfs, 2023.

[27] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Encryption policies for regulating access to outsourced data," *ACM TODS*, vol. 35, no. 2, April 2010.

[28] S. De Capitani di Vimercati, D. Facchinetti, S. Foresti, G. Oldani, S. Paraboschi, M. Rossi, and P. Samarati, "Multi-dimensional indexes for point and range queries on outsourced encrypted data," in *Proc. of GLOBECOM*, December 2021.

[29] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Keep a few: Outsourcing data while maintaining confidentiality," in *Proc. of ESORICS*, September 2009.