A dream come true: deletable content in immutable storage

VIKTOR TRÓN[*], Swarm Research Division, Switzerland

HENNING DIEDRICH, Ethersphere Core Task Force, KSCC, Germany

This work introduces DREAM, a novel construct that enables deletable content in otherwise immutable, decentralized storage systems such as Swarm. As equitable and feasable solution to the challenge of data destruction, the objective of DREAM is to reconcile the immanent technical permanence of content-addressed storage with the practical need and legal requirement of permanent revocability of access. The paper contributes a new deletion model that is based on access revocation and presents a mechanism that allows data-sharing users to grant and later rescind access to specific data to specific consumers. The solution leverages Swarm's distributed storage architecture to introduce a distributed one-time pad mechanism realized by a network protocol. The resulting system achieves the desired features of deniability, revocability, expirability, addressability, and malleability without requiring trusted intermediaries or complex cryptography. The approach preserves the censorship resistance of decentralized systems while allowing for sovereign control over data access. Security analysis shows that access revocation remains effective even under pervasive adversarial control of the network.

CCS Concepts: • **Networks** → **Network protocols**; • **Information systems** → **Information storage systems**; **World Wide Web**; • **Security and privacy** → **Database and storage security**.

Additional Key Words and Phrases: Swarm, Web3, decentralisation, data deletion, immutability

ACM Reference Format:
Viktor Trón and Henning Diedrich. 2025. A dream come true: deletable content in immutable storage. 1, 1 (April 2025), 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

This paper is structured as follows: Section 1 acknowledges the need for data deletion and explains why deletion is in practice today implemented as removal of access. It presents the challenges of enabling deletion in decentralized storage, exploring the tensions between censorship resistance and access control. Section 2 formalizes the concept of deletion in the context of content-addressed networks, like Swarm, and derives the criteria necessary for meaningful revocation of access. Section 3 describes the technical construction of the DREAM protocol, detailing its architecture, design rationale, and its five core properties. Section 4 analyzes the protocol's correctness and behavior under realistic network assumptions. Section 5 provides a security model and evaluates the system's robustness against adversarial behavior, especially the case of partial or pervasive network compromise. Section 6 concludes by summarizing the findings and highlighting the broader implications of user-friendly, access-controlled deletion in immutable storage environments.

[*]Corresponding author

Authors' addresses: Viktor Trón, viktor@ethswarm.org, Swarm Research Division, Neuchâtel, Switzerland; Henning Diedrich, hd@lexon.org, Ethersphere Core Task Force, KSCC, Berlin, Germany.

1 INTRODUCTION

The continuously increasing digitisation of data and the widespread use of cloud services have raised well-justified concerns regarding privacy and control of personal information to a new level [12]. In their responses, regulators often impose requirements on removability with little regard to technical and practical feasibility. Beyond the truism that it is virtually impsossible to make material once seen—i.e., downloaded to a user device—unseen, accepted legal and economic reasons effect that ostensibly 'deleted' data is often just hidden but physically preserved. Social media platforms list a plethora of exceptions to reserve the right to not actually delete the data of a deleted account [7]. While they are generally obliged by law to delete the data, some of their valid counter-arguments are themselves regulatory and legal requirements. Even down on the technical level, a local, high-performance database like Facebook's workhorse, Apache Cassandra, does not actually delete data [3] when it is flagged as 'deleted'—for the profane reason of higher performance, i.e., lower cost.

Entities that operate centralised infrastructure that enables digital publishing are usually in full control of the systems they use to serve content, so that they can carry out data "deletion" by simply *denying access* to it. This has practical advantages. But precisely the intention to take back this censorship power was the founding premise of *censorship-resistant*, decentralised network architectures. This purpose is a common denominator from the 1980 work of cypherpunks, up to their contemporary heirs, including the permissionless, distributed storage network, Swarm [9].

Centralised gatekeeping, however, must look appealing from a regulatory standpoint, as it can be coopted as an instrument of law enforcement and political censorship [10]. In case legal action is initiated, the law routinely forbids the destruction of data that could constitute evidence. So the gatekeeper becomes a well-defined responsible party with an effective, if paradoxical course of action: not serving censored content, but preserving it.

Even though this can give the unwary a false sense of privacy, it exemplifies the larger issue of censorship: publishing platforms possess the technological means to filter content, and centralised hosting makes it easy and economic to exert control over content. What in all likeliness started out as a benign,[1] sensible measure of content curation gradually transformed into extant censorship [5]. This is all the more problematic with social platforms [1, 6] that have gained quasi-monopolistic status as a result of network effects [14]. Due to the high costs associated with switching platforms, content creators increasingly find themselves 'deplatformed' [13] when they violate the arbitrary and changing publishing rules their platform of choice has adopted. Similarly, the ability to identify (central) hosts and coerce them into denying access to content through legal measures provides the powers that are with very economic means to curtail freedom of speech.

This is why, per its founding mission, in the decentralised paradigm of web 3, there is no longer a single operating entity controlling publishing platforms or hosting infrastructure. On the one hand, as intended, this renders the cost of censorship infeasible. Content once posted in a web 3 environment *can* disappear if it is not kept up, but the data is censorship-proof in that the disappearance cannot be hastened. On the other hand, the potential permanent exposure of personal data constitutes a significant concern for many individuals. Therefore, there is a need for novel solutions that can restore the sense of security that centralised gatekeepers represent [4].

Prima facie, the intent to introduce a mechanism to delete content from *immutable* platforms seems futile. But immutability does not mean that content cannot be updated, nor that the network itself never changes; just that what was stored once cannot *actively* be erased.

[1]David Chaum correctly predicted[2] the degrading effects of client-server architectures in the mid 80ies, kicking of the Cypherpunk movement that worked towards keeping electronic messaging and transacting censorship-free.

Accordingly, we are using the reality that 1) the requirement of data 'deletion' commonly does not mean physical destruction but the removal of access; combine this with the fact that 2) updates in immutable storage environments are performed by pointing to a newer version of data rather than physically replacing an old one; 3) data can be stored encrypted, so that losing access can be realized by 'losing the key;' 4) a deterministic random generator can be provided trustlessly, as a social process; 5) the act of an update can be used to change the seed of such deterministic random generator. Armed with these ingredients, we can arrive at a mechanism to *effectively* delete data from an immutable, distributed network, with this neutralising a perceived advantage of traditional, centralized data storage designs and comply with both user sentiment and legal requirements.

2 DELETION AND REVOKING ACCESS

Requirements of removal of information in the sense of erasing it from all physical storage devices is both unenforceable and impractical [8]. Even the most rigorous data protection audits do not require the erasure of offending data from backup tapes. In general, the tacit assumption is that information ordered to be removed should become *inaccessible* through *typical, original and precedented methods of access.*

In what follows, we formulate what we think is the strongest meaningful definition of deletion applicable to permissionless, immutable, decentralised storage systems and offer a construction that implements it. Importantly, this approach is purely technical. It focusses on the capabilities and costs of primary actors, rather than relying on procedural measures that impose obligations on intermediaries to respect the rights of primary actors. In other words, it works completely algorithmic in a trustless, distributed network without expecting anyone to comply.[2]

The primary actors in this context are users who wish to share content (the *uploaders*) by granting read access to a number of data consumers (*downloaders*). Granting read access is defined as providing a canonical *reference* to the content, allowing the system to retrieve the complete information that is intended to be disclosed.

Any party that is privileged to access this information could store, re-code, and potentially disseminate it, so that the content can be made accessible at a later point in time, which practically bypasses any process that would qualify as deletion (or removal of access) at the original download source. There is no protection possible against such adversity. Any legally and socially useful notion of deletion (e.g., revocation of access), therefore, defines a narrower case: *the viability to replay the same access method* at a lower cost than at least the full cost of storing the content.[3] At most, of storing all contents and all change logs of the publishing system holding the specifc content, because it might be unknowable what piece of information will become interesting at a later point in time.

Consequently, we will explore deletion as a scheme for uploading content with access revocation that meets the following criteria:

specialisation

The uploader is able to choose at the time of publishing a specialised construct that will allow revocation of access later. From a user's perspective, content that is meant to be reliably deleted should be marked as such at the time of upload. The cost of uploading such content may be higher than the cost of regular, censorship-resistant but non-deletable uploads.

sovereignty

Deletables presuppose access control, i.e., they are only available to a specific set of recipients. The uploader is the unconditional owner of the deletable data and is in possession of the means to selectively revoke access from any

[2]It even uses the entropy of the underlying, distributed network protocol itself to realize one of its components.

[3]That is, the total storage cost paid for the full size of the content starting from the time that access was revoked up till the attempted breach.

party that was previously granted access. In other words, the uploader's credentials are necessary to delete their own deletable content.

security

After the revocation of access, a grantee is unable to access the content using the same reference or any other cue shorter than the deleted content. I.e., if the downloader has *not* stored at least as much information as the deleted information itself, they will have no way of retrieving it.

Note that in the Swarm network, uploaded content may be *forgotten*: if nobody pays for storing it and the content is not accessed frequently, it will be garbage-collected. However, content no longer paid-for to be stored [4] cannot be regarded as reliably deleted: the requirements of sovereignty and security are not strictly fulfilled.

3 CONSTRUCTION

The goal of this section is to arrive at a formal construction of a revocable access model for the distributed storage network Swarm. We will base our exploration on *chunks*, the fundamental fixed-size storage units of Swarm's distributed storage model, DISC [11, §2]. The proposed *DREAM* construct implements a deletable content storage and access model that fulfills the requirements of specialisation, sovereignty, and security.

3.1 Dream: deniable, revocable, expirable, addressable, malleable

The use of the word 'dream' alludes to the counterintuitive nature of the finding that such a construct is even possible in an *immutable* model, as well as to the shifting and sometimes failing path across nodes that is its implementation. As mnemonic acronym, it resolves to the 5 *dream attributes* that express the above requirements of access control:

D *deniable* – The dream key serves as a one-time pad (OTP) for decryption. Since multiple content chunks (in fact, any arbitrary content) can use the same dream pad, the key's association to any specific content is plausibly deniable.

R *revocable* – Access granted through dream keys is revocable. Revoking access from all parties, including oneself, is considered deletion.

E *expirable* – The scheme allows for one-time use, i.e., in this design that the *key* can only be retrieved once.

A *addressable* – Access can be granted to nodes in the *neighbourhood* designated by *a*,[5] where only clients operating a node within a particular overlay address range are able to access the content.

M *malleable* – The construct is resilient to churn and dynamic changes in network size; it is reusable across independent grantees and upgradeable.

The dream protocol can be built on top of Swarm's DISC[6] APIs as a pure second-layer solution. Despite its rich feature set, the scheme does not use complex cryptography, but leverages the interplay of various component subsystems instead.

3.2 Chunk upload in Swarm

When content is uploaded to Swarm, the local client splits the data into 4-kilobyte chunks and pushes each chunk to the network using the *pushsync* protocol. This protocol is responsible for transmitting newly entered chunks to

[4]In the context of Swarm, chunks with *expired postage stamps*

[5]Swarm neighbourhoods are groups of nodes which are responsible for sharing the same chunks [9].

[6]DISC (Distributed Immutable Storage of Chunks) is a storage solution developed by Swarm based on a modified implementation of a Kademlia DHT which has been specialized for data storage. Swarm's implementation of a DHT differs in that it stores the content in the DHT directly, rather than just storing a list of seeders who are able to serve the content. This approach allows for significantly faster and more efficient retrieval of data.

the neighbourhood where they can be retrieved, i.e., the neighbourhood of nodes that holds the same content and is responsible for storing all the chunks sharing a prefix with the chunk's address.

A chunk is an association of a 32-byte address and a limited-length chunk payload. Let us define content-addressed chunk *CAC* as a function from bytes data with size limit of 4096 bytes and associated 8-byte metadata to chunks:[7]

$$\begin{aligned} \mathit{Chunks} &\overset{\text{def}}{=} \mathit{byte}\{32\} \times \mathit{byte}\{,4104\} && (1)\\ \mathit{CAC} &: \mathit{byte}\{,4096\} \times \mathit{byte}\{8\} \longrightarrow \mathit{Chunks} && (2)\\ \mathit{CAC}(p,m) &\overset{\text{def}}{=} \langle H(m \oplus p), m \oplus p\rangle && (3)\end{aligned}$$

The delivery of a new chunk to the node where it is to be stored can be seen as a request to which eventually the closest node responds with an acknowledgement of custody. The network protocols of Swarm use *forwarding Kademlia*, meaning that requests are routed to their destination through a series of hops successively relayed by *forwarding* nodes with ever increasing proximity to the target address.[8] The response is passed back via the same route as the original request (*backwarding*) keeping the identity of the request originator private, see figure 1.

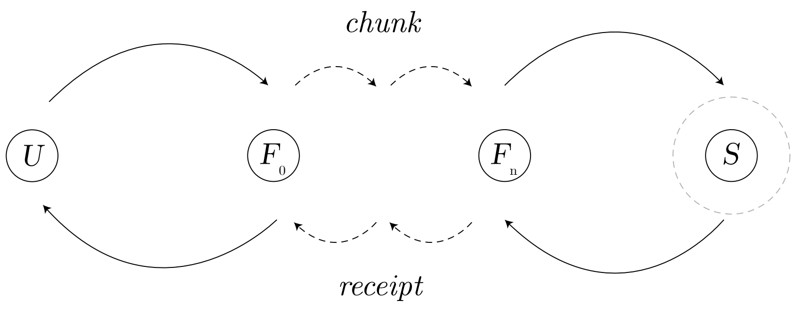

Fig. 1. Push syncing. The protocol is responsible for transferring newly uploaded chunks to the neighbourhood where they belong based on overlay address proximity to the content address. The length of the shared prefix prescribed is called storage depth. The protocol's request–response scheme uses forwarding/backwarding Kademlia relay.

This procedure requires spam protection to prevent mass injection of random chunks to the network that could expunge valuable chunks by triggering garbage collection on nodes that are pushed into capacity shortage. By requiring *postage stamps*, Swarm imposes an upfront cost to uploads as an effective measure against such spamming. Consequently, every chunk uploaded to Swarm is pushsynced and forwarded by relaying nodes towards its address where it is stored in the nodes' *reserves* provided it has a valid postage stamp attached.

3.3 High-level description

The central construct called a *dream pad* is a *chunk-sized one-time pad* (OTP), which acts as a decryption key for the deletable content. The dream pad is collectively created by a set of nodes following this protocol. The initial pad is a

[7]Content-addressed chunks attain their integrity by the fact that their address is deterministically derived from their content with the help of a one-way uniform digest, or *hash function* H. The hash function used to generate the content address of chunks in Swarm is based on a *binary merkle tree*, but this particular choice does not affect the correctness of our construction.

[8]Due to the Kademlia connectivity, the route to the node whose address is closest to the chunk can be found using local decisions by the nodes along the path. Also, the forwarding-backwarding routing scheme makes it possible to do bandwith accounting on a peer-to-peer basis and facilitates micropayment settlements on a repeated dealings basis with only a number of quasi-permanent peers that is logarithmic in the size of the network.

chunk-sized (4096-byte-long) sequence deterministically generated from a key-sized (32-byte) seed by a pseudo-random generator function $\mathcal{G}[4K]$.[9] The protocol starts with sending this pad as a chunk to the network. As per the pushsync protocol, a node that is closest to the chunk address is guaranteed to receive the chunk and is expected to update the pad and, wrapped as a chunk, send it to the network. The node that picks it up is considered the next node on the dream chain. When a node receives the dream chunk wrapping the input OTP, the node updates the dream chunk using data in its *reserve* (chunks that it is paid to store). If the protocol is initiated more than once, due to network and reserve changes, conditions may be different. However, as long as the initial pad is the same, the first chunk lands in the same neighbourhood and as long as the relevant parts of the reserve are unchanged, then it uses the same inputs and therefore ends up generating the same updated OTP. The same goes for every step on the way, until the grantee receives back in the n-the step the expected dream key, which then enables the decryption and reading of the deletable content. To lay it into the hands of the uploader that these conditions hold—or not—the pad is calculated using chunks from a batch under the control of the uploader.

This makes the retrieving of a deletable chunk a collective process involving multiple nodes. Calculating one part of the reference to a deletable chunk requires identifying a set of immutable chunks controlled by the uploader. The uploader's ability to change this underlying set provides the mutability to take access away.

3.4 Making up the dream

Let b be the ID (a 32-byte hash) of a postage batch owned by uploader U, and let $Chunks(\gamma, b, p)$ stand for the set of all chunks that belong to b (have a stamp issued under b) at blockheight γ in the bucket designated by the input chunk address a. Define $\overline{\chi}(\gamma, b, a)$ as the chunk in this set whose address is closest (has minimal XOR distance) to input chunk address a.

$$\overline{\chi} \quad : \quad Blockheight \times Batches \times Address \longrightarrow Chunks \tag{4}$$

$$\overline{\chi}(\gamma, b, a) \quad \overset{\text{def}}{=} \quad \underset{c \in Chunks(\gamma,b,a)}{\arg\min} \quad \chi(a, H(c)) \tag{5}$$

Let us now define the OTP update function Δ for input chunk c_i as follows:

$$\Delta \quad : \quad Chunks \longrightarrow Chunks \tag{6}$$

$$\Delta(c_i) \quad \overset{\text{def}}{=} \quad p_i{}^\wedge\overline{\chi}(\gamma_i, b, a_i) \text{ s.t.} \tag{7}$$

$$\gamma_i \text{ is the blockheight when } c_i \text{ arrives to the } i\text{-th node, and} \tag{8}$$

$$b \in B \text{ and } |B| = 1 \text{ for } B = \{b \in BatchIDs \mid b[0 : len(m_i)] = m_i \text{ where} \tag{9}$$

$$c_i = CAC(p_i, m_i) \text{ and } a_i = H(c_i) \tag{10}$$

If 1) the bucket of batch b designated by a_i is not empty,[10] and 2) the neighbourhoods include one or more buckets,[11] then the update function is locally computable by any storer node within the neighbourhood designated by a_i, as it will have all required input ($Chunks(\gamma, b, a_i)$) stored. Applying the update function to its own output, by the next node, we

[9]For example, the Keccak sponge function used throughout Ethereum for hashing does have this capability. Alternatively, the blockcipher encryption using the seed as initial nonce could be applied to a fixed constant chunk, such as all zeros.

[10]If batch b is underutilised, a bucket designated by a_i may not contain any chunks belonging to the batch bucket in which case the function is undefined.

[11]Since the chunks belonging to a bucket of batches all share a prefix that is longer than the prefix shared by the storer nodes in the neighbourhood, it is expected that all chunks in a batch bucket designated with a_i are stored in the reserve of every node in the relevant neighbourhood designated by a_i.

can define the *dream path* as follows:

$$\begin{aligned} \Pi \quad &: \quad Batches \times byte\{32\} \longrightarrow Chunks\{n\} && (11) \\ \Pi(b,g) \quad &\overset{\text{def}}{=} \quad c_0,\ldots,c_n Chunk\{n\} && (12) \\ c_i \quad &\overset{\text{def}}{=} \quad \begin{cases} CAC(\mathcal{G}[4K](g),\beta) & \text{if } i=0 \\ CAC(\Delta(c_{i-1}),\beta) & \text{otherwise} \end{cases} && (13) \\ & && (14) \end{aligned}$$

where $\beta = b[:8]$ is the 8-byte-long prefix of the batch ID b used as the metadata of the chunk sent.

Such dream path function Π is a pseudo-random generator with finite recursion. Finally, a *dream pair* is a particular pairing of a seed, the input to the generator function creating the initial pad p_0, and the dream pad, the output of n applications of the OTP update function.

Given a dream path of length n, a grantee overlay address a, and a batch b with a depth d, the uploader needs to find the generator g such that the n-th chunk of the dream path falls within a's neighbourhood (i.e., $PO(a, H(c_{n-1})) \geq d$).

$$\begin{aligned} dream \quad &: \quad Z^{+} \times Batches \longrightarrow \mathcal{P}(byte\{32\} \times OTP) && (15) \\ dream(n,d,b) \quad &\subset \quad byte\{32\} \times Chunks && (16) \\ \langle g,k\rangle \in dream(n,d,b) \quad &\Leftrightarrow \quad c_n = \Pi(b,g)[n] \wedge k = \textsc{Data}(c_n) \wedge PO(a,H(c_n)) \geq d && (17) \end{aligned}$$

Since $H(k)$ is uniform, the chance of $PO(a, H(k)) \geq d$ is 1 in 2^d. As a consequence, it is feasible for the uploader to find g, and from g to construct the initial pad data, which, in turn, allows them to calculate the dream pad k.

3.5 The Dream Protocol

The *dream* as a network protocol-based access method is *deniable, revocable, expirable, addressable, and malleable*. In order for grantees to calculate the dream pad, they must rely on the network, where each recursive step i of the calculation is performed by a node in the neighbourhood that is designated by the respective input chunk's address a_i.

In order to guarantee the correct termination, the following criteria must be fulfilled:

- all buckets of batch b designated by a_i for all $0 < i < n$ must be non-empty,
- the length of the prefix shared by the addresses of chunks in the same bucket must be greater than the storage depth d of the network,
- the chunk message c_i of each step $0 < i < n$ must be sent to the neighbourhood designated by $a_i = H(c_i)$,
- nodes at each step $0 < i < n$ of the dream path must both compute the OTP update and push output pad as a chunk to the network.

We define the dream network protocol (see figure 2): participants in the protocol listen to *dream chunks* (content address chunks whose metadata is a prefix of a valid batch ID b). The updated OTP is then wrapped together with β as a chunk message c_i and sent to the network address $H(c_i)$ as the next step in the dream chain. The receiving neighbourhood then calculates the next step based on this input. If the protocol is followed up to n steps, then the target node at address a receives c_n from which they can extract k.

The deletable dream chunk is constructed as follows: if uploader U wishes to grant downloader D (at overlay address a) revocable access to content chunk C, then U chooses a postage batch b that it owns, and that is *not completely*

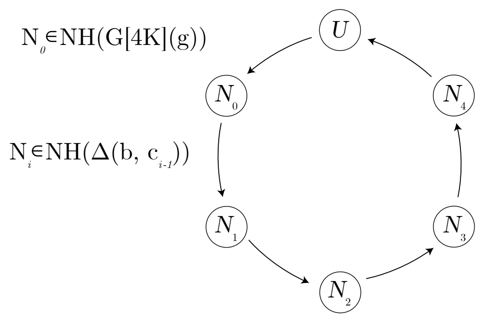

Fig. 2. The dream protocol. Assuming uploader mined a dream pair and Participants in the protocol listen to *dream chunks* (content address chunks whose metadata is a prefix of a valid batch ID b). They extract the OTP as the chunk data p_i, and also look up b using the prefix in the metadata m_i. They then find the chunk that are stamped with b, fall into their area of responsibility and whose address is closest to the input chunk's address $a_i = H(c_i)$. The updated OTP output p_{i+1} is then wrapped together with β as a chunk message c_{i+1} and sent to the network as the next step in the dream chain. The receiving neighbourhood then calculates the next step based on this input. If the protocol is followed up to n steps, then the target node at address a receives c_n from which they can extract k.

filled. U then by trying random seeds g, mines a dream pair $\langle g, k \rangle \in dream(n, d, b)$,[12] calculates the "ciphertext" $C' = C \wedge k$ and uploads it to Swarm, obtaining the parity reference $r = H(C')$. Now U creates the *dream chunk reference* $ref(C) = \langle r, b, g \rangle$, which can be given to D.

4 ANALYSIS

Downloader D, in possession of a dream chunk reference $ref(C) = \langle r, b, g \rangle$ calculated by U, constructs the initial pad $p_0 = \mathcal{G}[4K](g)$, wraps it as the initial chunk message $c_0 = CAC(p_0, b[:8])$, and sends it to Swarm. If and when D receives the dream chunk c back it extracts the dream pad as the chunk payload $k = \text{Data}(c)$.

D retrieves the parity chunk C' using reference r and then decodes the "plaintext" as $C = C' \wedge k$. The retrieval process is trivially correct as long as:

(1) The parity chunk C' is retrievable via normal retrieval methods. I.e., Swarm is up and running and the downloader can access it.
(2) The dream protocol is followed by all cooperating nodes.
(3) The contents of the batch bucket remains unchanged for all the neighbourhoods along the dream path.

We are now turning to access revocation, i.e., deleting content by making it inaccessible.

Uploader U had granted downloader D access to content C at address a through the dream reference $\langle r, b, g \rangle$. U *revokes* D*'s access by uploading extra chunks to batch* b. This changes $\overline{\chi}$ for some of the buckets of b along the dream path. As a result, downloader D will no longer be able to retrieve content C. Since the nearest chunk to the input's address a_i from batch b has changed for some $0 < i < n$, the dream path is diverted. This holds as long as there is at least one honest neighbourhood walked by the protocol. Because, if we assume that there is a bucket designated by some a_i on the dream path such that the $\overline{\chi}$ for the bucket has changed since the uploader constructed the dream; then when the participating node in the neighbourhood designated by a_i calculates the OTP update function, the result will

[12]We can assume that step-count n and depth d are security parameters.

be different than when the uploader granted the access and the dream chain will not terminate as it will return the wrong pad and be diverted in the wrong direction to find the next chunk.[13]

5 SECURITY

The reference to a dream chunk *does not leak* any information about the step-count or the neighbourhoods involved. Neither are the participants in the protocol aware of their position in the dream chain nor of any details about the other neighbourhoods that are part of it, except for the immediate next one to which they are pushsyncing their chunk.

The construction of the dream reference is *deniable*: if we consider our sensitive content C, and any uncontroversial content chunk A, then when producing C', the owner also produces $A' = A$^k and uploads it. When asked about k, producing A' makes the denial of other content, including C', more plausible.

As Swarm is a decentralized network, individual participants cannot (and should not) be trusted to play by the rules. The built-in incentives are what encourages cooperative behavior and helps maintain a Nash equilibrium. The necessary condition for content deletion is that in *at least one neighborhood* all nodes are honest, i.e., not all operators run malicious nodes which would store and serve information that they should not store and serve.[14]

A powerful adversary could potentially infiltrate every neighbourhood of Swarm and archive all information that has ever been uploaded (without being able to decipher it). It could also keep logs on what has been uploaded in what order, which would allow it to serve 'deleted' content. However, there is a huge cost associated with such indiscriminate archiving of all of Swarm's content, in what is the only way to reliably defeat the dream construction. Because the store would also contain all downloads, it is equivalent with the scenario where an adversary makes deletion redundant by keeping all downloaded data.

We now turn to the discussion of how to calibrate the step count in relation to the security model using the notion of *network-wide neighbourhood infiltration rate*. For instance, a rate of $\frac{1}{2}$ means that the chance of infiltration, in any neighbourhood independently and uniformly across the network, is 1 in 2, implying that, on average, we can expect half of the neighbourhoods in the network to contain a malicious and colluding node.

For access to be revoked, the owner uploads new chunks in each neighbourhood along the dream path. We assume that malicious nodes are able and willing to disregard these new chunks and, in an attempt to render the revocation of access impotent, respond to a request the same as before the revocation. If each neighbourhood in the path contains at least one malicious node, it is, in principle, possible that the dream protocol gives the unchanged response, and access is granted against the intention of the uploader, i.e., the protocol terminates allowing access even though it was revoked. Given a specific dream chain, however, if even a single neighbourhood on the dream path is honest, they will respect the chunks that newly arrived since the construction and therefore, divert the dream path, preventing it from terminating with the grantee.

Thus, for a breach of access to happen, all neighbourhoods must be malicious. In our security model, a neighbourhood is malicious with uniform and independent probability. For an overall infiltration rate of 1 out of k, the chance of all neighbourhoods on a given random dream path being malicious is k^{-n}. For a security requirement of a success rate of σ "nines", i.e., for an error rate less than $10^{-\sigma}$, we can formulate the requirement as

$$k^{-n} \leq 10^{-\sigma} \tag{18}$$

[13]Since we cannot know which nodes are colluding and malicious, the uploader needs to upload batch bucket changes to all the neighbourhoods on the dream path to reliably revoke access.

[14]With the growth of the Swarm network, the number of neighbourhoods increases whereas the membership size per neighborhoods remains about the same. Thus, the costs of attacking the proposed protocol increases with network size.

Now, expressing k as a power of 10, $k = 10^{\kappa}$, taking the logarithm of both sides and multiplying by -1, we get

$$n \geq \frac{\sigma}{\kappa} \tag{19}$$

As an example, with one in every 10 neighbourhoods being malicious ($\kappa = 1$), we calibrate $n \geq \sigma$: the dream path must be as long as the number of nines expressing the success rate.[15]

6 CONCLUSION

This paper presents a practical mechanism, dream, to achieve deletable content in immutable, decentralized storage, based on the concept of revocable access. By formalizing deletion as the loss of access rather than the physical erasure of data, the dream construct offers a secure, permissionless, and self-sovereign solution. The system is designed to integrate seamlessly with Swarm's storage model, DISC, and relies on simple, yet robust protocol interactions among peer nodes. Through the introduction of a distributed one-time pad and a network-driven computation model, dream enables users to selectively revoke access to content without undermining the foundational principles of decentralization and censorship resistance. The result is a useful balance between data permanence and sovereign control, offering a much-needed, viable approach to privacy-conscious publishing in the Web3 era.

ACKNOWLEDGMENTS

The authors would like to thank György Barabás, Andrea Robert for help with text and formatting, Daniel Nickless for the figures and members of the Swarm Research Division, especially Daniel A. Nagy for their ideas leading to this paper.

REFERENCES

[1] Kyle Langvardt Barnett. 2019. Platform or Publisher? The legal classification of social media. *Fordham Intellectual Property, Media & Entertainment Law Journal* 29, 2 (2019).

[2] David Chaum. 1984. A New Paradigm for Individuals in the Information Age. https://chaum.com/wp-content/uploads/2022/01/Chaum-New-Paradigm-for-Individuals-in-Information-Age.pdf

[3] The Apache Software Foundation. 2025. Cassandra Documentation/Tombstones. https://cassandra.apache.org/doc/5.0/cassandra/managing/operating/compaction/tombstones.html#deletes-without-tombstones

[4] Roxana Geambasu, Tadayoshi Kohno, Arvind Krishnamurthy Levy, and Henry M Levy. 2009. Vanish: Increasing data privacy with self-destructing data. In *USENIX Security Symposium*. USENIX Association, USENIX Association, 1–10.

[5] Tarleton Gillespie. 2018. *Custodians of the Internet: Platforms, content moderation, and the hidden decisions that shape social media.* Yale University Press, New Haven, CT.

[6] Kate Klonick. 2018. The New Governors: The People, Rules, and Processes Governing Online Speech. *Harvard Law Review* 131 (2018), 1598–1670.

[7] Inc. Meta Platforms. 2025. Privacy Policy, How Long Do We Keep Your Information? https://www.facebook.com/privacy/policy/?section_id=8-HowLongDoWe

[8] Paul Ohm. 2010. Broken promises of privacy: Responding to the surprising failure of anonymization. *UCLA Law Review* 57 (2010), 1701–1777.

[9] The Swarm Team. 2021. Swarm. https://www.ethswarm.org/swarm-whitepaper.pdf

[10] Ryan Tracy. 2023. Facebook Bowed to White House Pressure, Removed Covid Posts. https://www.wsj.com/politics/policy/facebook-bowed-to-white-house-pressure-removed-covid-posts-2df436b7?mod=article_inline

[11] Viktor Trón. 2025. The Book of Swarm. https://www.ethswarm.org/the-book-of-swarm-2.pdf. printed edition, Jan 2025.

[12] Paul Voigt and Axel Von dem Bussche. 2017. The EU General Data Protection Regulation (GDPR). https://doi.org/10.1007/978-3-319-57959-7

[13] Alice Zhang et al. 2021. Deplatforming: Following extreme internet celebrities to Telegram and alternative social media. *First Monday* 26, 2 (2021). https://doi.org/10.5210/fm.v26i2.10630

[14] Ethan Zuckerman. 2014. New media, new civics? *Policy & Internet* 6, 2 (2014), 151–168.

[15] a 6-hop-long path will guarantee access revocation with a certainty of 99.9999%.