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, decentralised storage systems such as Swarm. As an equitable and feasible response to the challenge of data destruction, DREAM reconciles the inherent persistence of content-addressed storage with the practical needs, user preferences, and legal requirements of permanent access revocation. The paper contributes a new deletion model based on access revocation, and presents a mechanism that allows data-sharing users to grant—and later rescind—access to specific data for specific consumers. The solution leverages Swarm's distributed storage architecture to introduce a procedural way to retrieve decryption keys, realised by a network protocol. The resulting system achieves the desired properties of deniability, revocability, expirability, addressability, and malleability without requiring trusted intermediaries or complex cryptographic primitives. The approach preserves the censorship resistance of decentralised systems allowing for sovereign control over data access by the original uploader, but not by any other party, thus avoiding any re-centralisation in order to achieve deletion. Security analysis shows that access revocation remains effective even under pervasive adversarial control.

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 (September 2025), 14 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, in practice, deletion is typically implemented and understood as removal of access. It presents the challenges of realising deletion in decentralised storage systems, exploring the tension between censorship resistance and access control. Section 2 formalises the concept of deletion in the context of content-addressed networks like Swarm and derives the criteria required for meaningful access revocation. Section 3 describes the technical construction of the DREAM protocol, detailing its architecture, design rationale, and its five core properties. Section 4 analyses the protocol's correctness and behaviour under realistic network assumptions. Section 5 provides a security model and evaluates the system's resilience against adversarial behaviour, in particular, under conditions of pervasive network compromise. Section 6 discusses the prerequisites for implementing DREAM and argues that adopting it would require only modest effort. Section 7 concludes by summarising the findings and highlighting the broader implications of user-friendly, access-controlled deletion in the context of immutable storage.

*Corresponding author

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

Manuscript submitted to ACM

1 Introduction

The continuously increasing digitisation of data and the widespread use of cloud services in recent years have raised well-justified concerns regarding privacy and control of personal information [13]. In response, regulators often impose requirements for data removability with little regard for technical or practical feasibility. Beyond the truism that it is virtually impossible to make material once seen unseen, legal and economic realities often mean that ostensibly 'deleted' data is merely hidden rather than physically erased. Social media platforms, for example, list a plethora of exceptions to reserve the right to not actually delete the data of a 'deleted' account [8]. And while they are generally obliged by law to erase the data, some of their valid counter-arguments are themselves regulatory and legal requirements. Even at 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,' purely for the mundane reason of higher performance, i.e. lower cost.

But because entities that operate centralised infrastructure for digital publishing typically have full control over the systems that serve their content, they can carry out the effective 'deletion' of data by simply *denying access* to it.

No wonder then that centralised gatekeeping is appealing from a regulatory standpoint: it can be summoned as a convenient instrument of law enforcement. And sure enough, in case legal action is initiated, the law routinely forbids the destruction of data that could constitute evidence. The centralized gatekeeper is thus morphed into a 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 illustrates a deeper issue with censorship: publishing platforms possess the technological means to filter content, and centralised hosting makes it easy and economical to exert control over it. What may have started out as a benign and sensible measure of content curation can gradually transform into extant censorship [2, 5]. This is all the more problematic with social platforms [1, 6] that have gained quasi-monopolistic status as a result of network effects [15]. This allowed governments to co-opt moderation tools in service of political agendas [11]. And due to the high costs associated with switching platforms, content creators increasingly find themselves 'deplatformed' [14] for violating the arbitrary and frequently changing publishing rules of the platforms they rely on. The ability to identify (central) hosts and coerce them through legal means to deny access to content provides authorities with a cheap and easy tool to curtail freedom of speech.

Recognising how easily technically motivated solutions can be put in service of propaganda, manipulation and surveillance has motivated the work of hackers for decades. The vision to enable censorship-resistant publishing has been at the forefront of the decentralisation efforts they initiated. Since the early days of the *cypherpunk movement* in the 1980s, *censorship resistance* has become a foundational principle of peer-to-peer storage networks, such as Swarm [10].

In the decentralised paradigm of Web3, there is, as per its founding mission, no longer a single operating entity controlling publishing platforms or hosting infrastructure. On the one hand, this renders censorship[1] infeasibly costly, as intended. Content posted in a Web3 environment *can* disappear if it is not kept up. But the data is censorship-proof in that its disappearance cannot be hastened by anyone other than the original poster. With a decentralized, immutable platform, even the poster cannot make data disappear at will though. And thus, the potentially unintended yet permanent exposure of personal data presents a serious concern for many individuals. Unless one accepts this as the price to be paid for censorship resistence, there is a need for solutions that can restore the sense of security that centralised gatekeepers provide [4].

[1] Censorship is understood as the act of a third party to make data unavailable. In practice, this is the case when said 'single operating entity controlling publishing platforms or hosting infrastructure' makes content inaccessible. Censorship is not the deletion of something previously posted by its original poster or uploader, which, when enabled in whatever form, does not therefore violate censorship-resistance.

Prima facie, the intent to introduce a mechanism to delete content from *immutable* platforms seems futile. However, immutability does not imply that content cannot be updated, nor that the network itself cannot change; it simply means that once data is stored, it cannot be *actively* deleted.

To overcome this apparent paradox, we need to acknowledge that 1) the requirement of 'deletion' normally does not refer to physical destruction of offending content but to the removal of access to it; combine this with the fact that 2) data can be stored encrypted, making loss of access equivalent to 'losing the key'; that 3) a deterministic random generator can be provided trustlessly to create a key through a social process defined by a network protocol, as explained below; that 4) immutable storage environments allow for adding data; and 5) that the act of adding seemingly unrelated content can change the protocol's output, altering the key derived from the same seed.

Armed with these ingredients, we can arrive at a mechanism to *effectively* delete data from an immutable, distributed network, and comply with both user sentiment and legal requirements; and with this, neutralise a perceived advantage of traditional, centralised data storage designs.

2 Deletion and revoking access

Requirements of removal of information in the sense of erasing it from all physical storage devices are both unenforceable and impractical [9]. 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 relies on the capabilities and costs of primary actors, rather than on procedural measures that impose obligations on intermediaries to respect those actors' rights. In other words, it operates completely algorithmically in a trustless, distributed network, without expecting anyone to comply either out of altruistic motive or under punitive threat.

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 intended for disclosure. Revoking access means that this canonical reference stops working.

It is obvious that any party that is privileged to access 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). As there is no guaranteed protection against such adversity, any legally and socially useful notion of deletion invariably defines a narrower case: taking away *the viability to replay the same access method* at a lower cost than at least the full cost of storing the content.[2] At most, this could mean the full cost of storing all content and/or all change logs of the publishing system holding the specific content, since it is impossible to know in advance which piece of data may become relevant in the future. For example, deletion on a social media platform only means that the original link no longer works; even if others may have downloaded and reposted the content elsewhere under a new link.

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

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

specialisation

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

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 holds the exclusive means to selectively revoke access from any party that was previously granted access. In other words, the uploader's credentials are necessary to delete their own deletable content, and no one else can delete it.

security

After access is revoked, a grantee is unable to access the content using the same reference or any other cue shorter than the deleted content itself. That is, unless the downloader has stored at least as much as the size of the deleted content itself, they will have no way of retrieving it.

Note that in the Swarm network, uploaded content may eventually be *forgotten*: if nobody pays for its storage and the content is not frequently accessed, it will be garbage-collected. However, content no longer paid for to be stored [3] 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 [12, §2]. The proposed *DREAM* construct implements a deletable content storage and access model that fulfills the requirements of specialisation, sovereignty, and security.

3.1 Synopsis

The general approach of the contribution is that access to content can be revoked by taking away the decryption key. For this to be possible, the key itself must be of volatile nature. In the proposed protocol, such a key (k) is constructed for the grantee by data flowing through the network of Swarm nodes (see figure 2). Starting out as a chunk-length piece of data generated from a random seed (g) it is sent to its destination wrapped as a content-addressed chunk beginning its journey of transformation following a *dream path*. At each step, the chunk is picked up by a storer node in the target neighbourhood, the content is modulated using local chunk data in the node's storage as input (Δ function) then once-again wrapped as a chunk, its content address determining in which neighbourhood the subsequent step will be performed. After an arbitrary but fixed number of steps (n), the dream path leads back to the grantee and the key k is extracted and can be used to reconstruct the deletable content C by decrypting its encrypted form C' that is stored in the network with k.

The uploader of deletable content does not need to play this script of collective construction: critically, because each step takes input from data previously uploaded using the same *postage stamp* owned by the uploader, they have all the data needed to simulate the protocol's calculations. Once the owner mines a key by finding a path, the deletable content (C) is encrypted, and the resulting encrypted content (C') is then uploaded. To share a link to the content,

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

the uploader shares a dream reference, a tuple $\langle r, b, g \rangle$ consisting of the Swarm reference r to the encrypted content C', a postage stamp batch reference to the batch ID b, and the generator seed g. The grantee retrieves the encrypted content by the Swarm reference r. The batch id b and the generator seed g enable the grantee to obtain the key from the network, triggering the path traversal to re-create the dream key k. The content plaintext C can be obtained by decrypting C' with the key k. Since the key itself has the same length as the content, there is no incentive to store $\langle r, k \rangle$ as an alternative, because it would take more storage than simply storing the content C.

Importantly, while the key to the data can be accessed through the network, it is never stored in it. This is what makes revocation possible. Since the owner of the stamp (the uploader) controls that dataset, by virtue of uploading extra data to any node's local storage, they can change the spice coming from local data (figure 3). This is sufficient to deflect the path which renders the key inaccessible, and thus effectively delete their content from the network.

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

The use of the word 'dream' alludes to the counterintuitive nature of the finding that deletion is even possible in an *immutable* context. As a mnemonic acronym, it resolves to the 5 *dream attributes* that the proposed construct displays:

D *deniable*
: The dream key constructed by the protocol is used for both encryption and decryption. Since any content chunk (in fact, any arbitrary content) could be encrypted using it, the key's association to any specific content is plausibly deniable (see section 5).

R *revocable*
: Access granted through dream keys is revocable. Revoking access from all parties, including oneself, is considered deletion. What is actually revoked is the ability to retrieve the key, which invalidates the reference $\langle r, b, g \rangle$ to the decrypted content.

E *expirable*
: The scheme allows for one-time use in this design. By strategically limiting the Swarm postage stamp supply associated with the key for its (re-)creation, the *key* can be made to only be retrievable once.

A *addressable*
: Access can be granted to nodes in any *neighbourhood* h by mining a dream path of length n such that the last chunk falls in h,[4] 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[5] APIs as a pure second-layer solution. Despite its rich feature set, the scheme does not use complex cryptographic primitives, but instead leverages the interplay of various component subsystems.

[4] Swarm neighbourhoods are groups of nodes which are responsible for sharing the same chunks [10].
[5] DISC (Distributed Immutable Storage of Chunks) is a storage solution developed by Swarm based on a modified interpretation of a Kademlia DHT (Distributed Hash Table) 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.

3.3 Chunk upload in Swarm

When content is uploaded to Swarm, as per its standard operation, the local Swarm client splits the data into 4-kilobyte fragments, wraps each of them with some metadata and an address into a *chunk*, and pushes them to the network using the *push-sync* protocol. This protocol governs the transmission of newly entered chunks to the neighbourhood where they are to be stored.

A Swarm chunk c is an association $\langle a, c' \rangle$ of a 32-byte address a and chunk content c'. While Swarm has other chunk types, the DREAM protocol focuses on the *content-addressed chunk* (CAC). CACs assume that their content c' is the chunk data p of length limited to 4096 bytes prepended with associated metadata m. Content-addressed chunks attain their integrity by having their address deterministically derived from their content (including metadata and chunk data) with the help of a one-way uniform digest, a *hash function* H.[6]

$$Data \stackrel{\text{def}}{=} byte\{, 4096\} \tag{1}$$

$$Meta \stackrel{\text{def}}{=} byte\{8\} \tag{2}$$

$$Address \stackrel{\text{def}}{=} byte\{32\} \tag{3}$$

$$Content \stackrel{\text{def}}{=} Data \times Meta \tag{4}$$

$$Chunk \stackrel{\text{def}}{=} Address \times Content \tag{5}$$

$$CAC : Data \times Meta \to Chunk \tag{6}$$

$$CAC(p, m) \stackrel{\text{def}}{=} \langle Keccak(m \oplus BMT - root(p)), m \oplus p \rangle \tag{7}$$

We also define:

$$Address : Chunk \to Address \tag{8}$$

$$Address(\langle a, c' \rangle) \stackrel{\text{def}}{=} a \tag{9}$$

$$Data : Chunk \to Data \tag{10}$$

$$Data(\langle a, c' \rangle) \stackrel{\text{def}}{=} c'[8:] \tag{11}$$

The delivery of a new chunk to the node where it is to be stored can be seen as a request, to which one of the nodes closest to the chunk's hash position in the DHT eventually responds with an acknowledgment of custody. Swarm's network protocols use *forwarding Kademlia*, meaning that requests are routed to their destination through a series of hops successively relayed by the *forwarding* nodes with ever increasing proximity to the target address, i.e. the chunk's address a. 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.

Proximity is the inverse of the xor distance metric [7], while *proximity order* $PO(x, y)$ returns the integer part of the logarithm of the xor distance, which can be calculated as the number of matching initial bits of the two addresses x and y.[7] Both proximity and proximity order apply to the address space shared by chunks and peers and play a crucial role in Swarm's peer-to-peer routing. Conversely, an address range is specified by a bit sequence, which defines a *neighbourhood* of chunks (reserve), or peers. Given an address a and a non-negative integer depth d, the shared *prefix*

[6] The BMT hash is the Keccak256 hash of the metadata prepended to the root hash of the *binary Merkle tree* with a Keccak256 base hash over 32-byte segments of the chunk data. This particular choice of H does not affect the correctness of our construction.

[7] As an example, the proximity order PO of 00101100 and 00110011 is 3, because the two 8-bit integers share the prefix 001, which is three bits long.

of all addresses in the range can be specified as the initial d bits of a, called a neighbourhood of *depth d designated by a*. In practice, 'neighbourhood' refers to a cluster of nodes that are incentivised to relay as well as replicate and retain all chunks whose addresses share such prefix with the addresses of nodes in the cluster.[8]

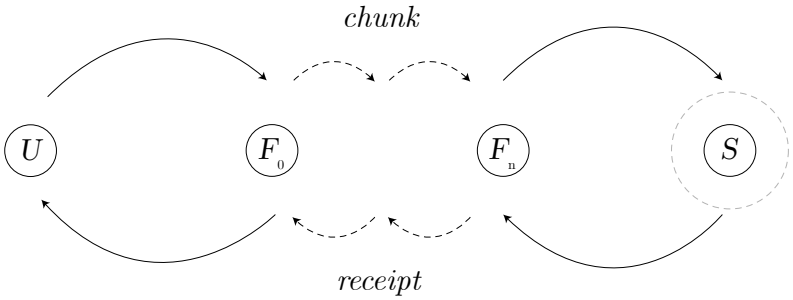

Fig. 1. Push-syncing. The protocol is responsible for transferring newly uploaded chunks to the neighbourhood of peers where they belong based on address proximity to the chunks' content addresses. Uploader U posts a *chunk* to a local peer connection F_0, which forwards it to a node closer to storer S, etc. The protocol's request—response scheme uses the forwarding/backwarding Kademlia relay.

Eventually, every chunk uploaded to Swarm is push-synced and forwarded by relaying nodes toward its address, where it is stored in the nodes' *reserve*. The length of the shared prefix prescribed to accept a neighbourhood as target area is called storage depth $depth_t$. It depends on the amount of storage space nodes must dedicate for their reserve and the overall volume uploaded to the network. The depth is incremented as the volume doubles and decremented as the volume halves.

This procedure requires spam protection to prevent mass injection of random chunks into the network, which 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. A chunk is accepted into a node's reserve only if it has a valid postage stamp attached. Revenue resulting from postage stamp purchases constitute the reward pot for the redistribution game, compensating storage nodes for data retention.

3.4 High-level description

The central construct of this contribution is called the *dream key* (k). It is a random byte sequence the size of a chunk's data (p), and it acts as a symmetric key (for encryption and decryption) for the deletable content C. Revoking access to the dream key k realises the deletion of the content C that was encrypted using it. The critical question we are addressing is how access could be revoked in an immutable storage network. The solution is to never store the dream key directly, but to have it collectively created by a set of nodes on demand, according to the protocol described below.[9]

The creation of the key starts with a chunk-sized (4096-byte) sequence p_0 deterministically generated from a key-sized (32-byte) seed g by a pseudo-random generator function $\mathcal{G}[4K]$.[10] The protocol then sends this first iteration stage

[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 bandwidth accounting on a peer-to-peer basis and facilitates micropayment settlements on a repeated-dealings basis with a subset of quasi-permanent peers that is logarithmic in the size of the network.

[9] As mentioned, since the dream key is as large as the data it encrypts, there is no advantage for anyone to store it instead of the decrypted data itself.

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

of the key as a content-addressed chunk (CAC) $c_0 = \langle a_0, m \oplus p_0 \rangle$ to the network, m being metadata that helps with the key generation. As per Swarm's push-sync protocol, a node v_0 in the neighbourhood designated by the chunk address a_0 is guaranteed to eventually receive this chunk. Now, the dream protocol prescribes that the closest node must update the key p_0 to p_1 in a specific way, using b and the *key update function* Δ; and then, wrapped as a new chunk $c_1 = \langle a_1, m \oplus p_1 \rangle$, sent it to the network again towards the neighbourhood designated by its new content address a_1. The next node that updates the dream chunk is considered the next node on the dream path ($v_0 \dots v_n$ in figure 2).[11]

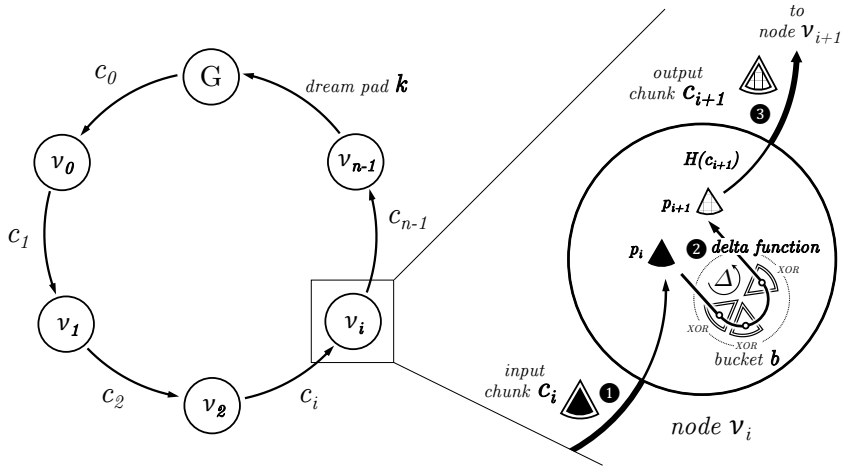

Fig. 2. Left: The path of the solid arrows represents the dream path, a series of neighbourhoods the dream protocol needs to traverse. The grantee posts the chunk data p_0 generated from g wrapped as chunk $c_0 = \langle a_0, m \oplus p_0 \rangle$ no different from any other upload to Swarm. Through multiple forwarding hops, c_0 arrives where it would be stored according to $a_0 = Address(c_0)$. The dream key is created as the chunk-long piece of data journeys through the neighbourhoods designated by $Address(c_0), \dots, Address(c_n)$, picked up by the closest nodes $v_0 \dots v_n$. Eventually, at step $n+1$, c_n is received back by grantee G at address a who can extract the dream key $k = Data(c_n)$. Right: For each step $0 \le i < n$ on the way node v_i updates c_i using the update function Δ. Information locally available to nodes specific to that neighbourhood ($BB_t(b, c_i)$) serves as input to the key update function (Δ). The (*1.*) incoming chunk c_i is used to (*2.*) find in the reserve the set $BB_t(b, c_i)$, i.e. those chunks that have a valid postage stamp issued under b and fall into the batch bucket designated by the input chunk address $Address(c_i)$. These chunks' data are then XOR-ed together with the input chunk data $Data(c_i)$ to create p_{i+1}, which is then wrapped into the chunk c_{i+1} and uploaded (*3.*).

Every node that receives the evolving dream chunk wraps it using data from its *reserve* at the time (BB_t) as input for Δ, i.e. from the chunks that it is paid to store. If the protocol is initiated more than once, conditions may be different due to network and reserve changes. This sensitivity of conditions to time is indicated with a subscript t. In fact, the key invention is to use the ability of the uploader to change the reserves of a neighbourhood, to thereby derail a dream path that previously resolved.

But as long as the initial key p_0 and the metadata m are the same and the relevant parts of the reserve BB remain unchanged, the system ends up generating the same dream key, which allows the grantee to decrypt and read the deletable content.

Critically, by leaving control over these conditions in the hands of the uploader, they are the only party that can actively block the grantee's ability to calculate the key. Overall, this makes retrieving a deletable chunk a process

[11] Note that, due to forwarding Kademlia, routing is realised by relaying the messages via multiple hops using keep-alive peer connections as the transmission channel. However, for the dream protocol, these intermediate forwarding nodes are not relevant.

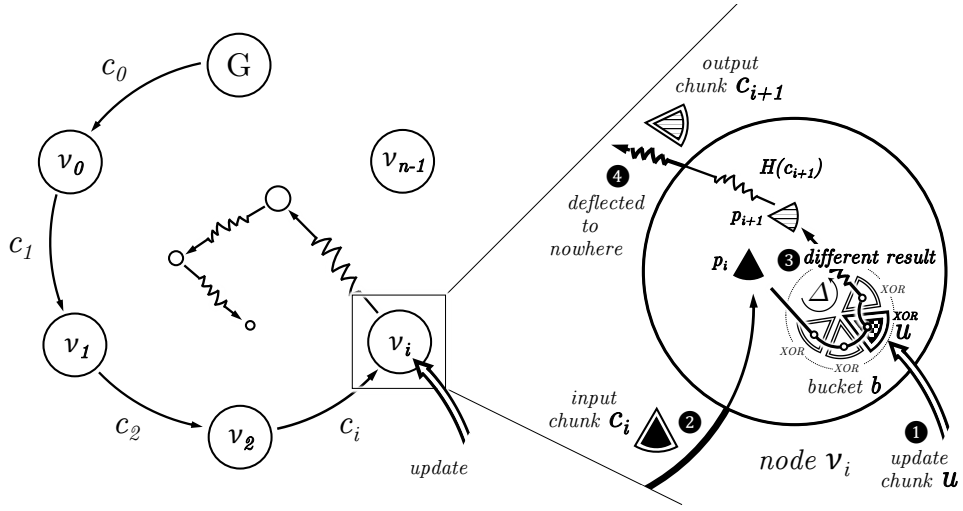

Fig. 3. Deletion. The dream path is shown as 'deflected' by a new chunk u stamped by postage batch b sent by uploader U (*1.*). An uploader may alter the dream path by inserting a chunk at time t' into the relevant batch bucket in any step $0 \leq i < n$. As a result, $BB_t(b, c_i) \neq BB_{t''}(b, c_i)$ for all $t < t' < t''$ for at least those steps i effected. This change will result in a different output chunk c_{i+1} at step i, whereby the dream path is *deflected* and will not terminate with the grantee who is now unable to retrieve the dream key to access the 'deleted' content.

involving multiple nodes. Because calculating one part of the reference to a deletable chunk requires sorting through a set of immutable chunks controlled by the uploader, the uploader's ability to change this underlying set—by adding to it—provides the mutability to take access away (figure 3).

3.5 Making up the dream

The specific way that each node in the dream path wraps the chunk that will become the dream key before sending it on is described in the following (see figure 2). Let the 32-byte hash b be the ID of a postage batch, i.e. a set of stamps, owned by uploader U. The node receiving a dream chunk looks b up in their batchstore by the prefix extracted from the chunk metadata. We assume that this prefix will always match b's ID uniquely, as long as b is valid at the time t of receiving the chunk. Chunks in a bucket share a prefix corresponding to the *uniformity depth* $v(b)$. In order to ensure that nodes can always detect doubly signed storage slots, Swarm stipulates that neighbourhood depth must not exceed uniformity depth for the entire period that the stamp is valid:

$$\forall b \in Batch, \forall t, b \text{ is valid at } t \longrightarrow v(b) \geq depth_t \tag{12}$$

Let $BB_t(b, a)$ represent the set of chunks belonging to batch b at time t and fall into the bucket designated by a. Since the chunks belonging to a bucket of a batch all share a prefix longer than the prefix shared by the storer nodes in the neighbourhood, it is expected that all chunks in a batch bucket designated by a are stored in the reserves of nodes in the relevant neighbourhood designated by a. It follows from equation 12 that neighbourhoods' reserves must contain entire buckets:

$$\forall a \in Address, b \in Batch, \forall t, b \text{ is valid at } t \longrightarrow BB_t(b, a) \subseteq \{c \mid PO(v, c) \geq depth_t\} \tag{13}$$

The *key update function* Δ updates the key by bitwise XOR-ing it with each chunk data belonging to batch b at block height t and falling into the bucket designated by the input chunk's address.

$$\Delta \quad : \quad Batch \times Chunk \rightarrow Data \tag{14}$$

$$\Delta(b, \langle a_i, c_i \rangle) \quad : \quad Data(c_i) \veebar \bigvee_{c \in BB_t(b, a_i)} Data(c) \tag{15}$$

If the bucket of batch b designated by a_i is empty, then Δ returns $Data(c_i)$. In any case, the update function is locally computable by any storer node within the neighbourhood designated by a_i, as it will have all required input (reserve chunks in $BB_t(b, a_i)$) stored.

Applying the update function Δ to its own output, by the next node, we can define the *dream path* Π as follows:

$$\Pi \quad : \quad Batch \times Address \rightarrow Chunk\{n+1\} \tag{16}$$

$$\Pi(b, g) \quad \stackrel{\text{def}}{=} \quad c_0, \ldots, c_n \text{ such that} \tag{17}$$

$$c_i \quad \stackrel{\text{def}}{=} \quad CAC(p, b[0:8]) \text{ where } p = \begin{cases} \mathcal{G}[4K](g) & \text{if } i = 0 \\ \Delta(b, c_{i-1}) & \text{otherwise} \end{cases} \tag{18}$$

Finally, a *dream pair* $\langle g, p_n \rangle$ is a particular pairing of a seed g that is the input to the generator function creating the initial key p_0; and the dream key p_n, the result of n applications of the key update function.

The starting point, seed g, such that the dream path ends after n steps in a neighbourhood a of nodes that U desires to be the potential exclusive receivers of the deletable chunk's payload, cannot be calculated directly. It is instead worked out by trial-and-error, or in other words, g is mined.

Therefore, given a dream path of length n, a grantee overlay address a, and a batch b, the uploader needs to find the generator g such that the last chunk of the dream path falls within a's neighbourhood.

$$dream \quad : \quad Address \times Batch \times \mathbb{Z}^+ \rightarrow \mathcal{P}(Address \times Data) \tag{19}$$

$$dream(a, b, n) \quad \subset \quad Address \times Data \tag{20}$$

$$\langle g, k \rangle \in dream(a, b, n) \quad \Leftrightarrow \quad k = Data(c_n) \wedge c_n = \Pi(b, g)[n+1] \wedge PO(a, Address(c_n)) \geq d \tag{21}$$

Since the hash function H used in calculating $a_n = Address(c_n)$ is uniform, the chance of $PO_{256}(a, a_n) \geq d$ is 1 in 2^d. As a consequence, it is feasible for the uploader to find g for any sufficiently low depth d, which, in turn, allows them to calculate the dream key k for a dream path ending at a.

3.6 The Dream Protocol

To calculate the dream key, grantees must rely on the network, where each recursive step i of the calculation for all $0 \leq i \leq n$ must be performed by the node closest to the 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 $0 \leq i < n$ 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 c_i of each step $0 \leq i \leq n$ must be sent to the neighbourhood designated by $a_i = Address(c_i)$,
- nodes at each step $0 \leq i < n$ of the dream path must both compute the key update and push the output key as a chunk to the network.

We define the dream network protocol as follows: Assuming uploader U mined a *dream pair* $\langle g, k\rangle$ of length n for grantee at overlay a and nodes in the network listen to *dream chunks*, content-addressed chunks whose metadata β is a prefix of a valid batch ID b. Once they receive dream chunk c_i, and also look up b using prefix β, they then find the chunks that are stamped with b, fall into the bucket of the input chunk address a_i. The updated key output $k_{i+1} = \Delta(b, c_i)$ is then wrapped together with β as a content-addressed chunk $c_{i+1} = CAC(k_{i+1}, \beta)$ and uploaded to the network to end up in the neighbourhood designated by $a_{i+1} = Address(c_{i+1})$ as the next step on the dream path. The node that is closest to a_{i+1} 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 it can extract $k = Data(c_n)$.

The deletable dream chunk is constructed as follows: if uploader U wishes to grant downloader D (at overlay address a) revocable access to chunk content C, then U chooses a postage batch b that it owns, and that is *not completely filled*. U then by trying random seeds g, mines a dream pair $\langle g, k\rangle \in dream(a, b, n)$, calculates the ciphertext $C' = C \veebar k$ 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 must be privately shared with grantee D.

4 Analysis

Downloader D, in possession of a dream chunk reference $ref(C) = \langle r, b, g\rangle$ calculated by U, constructs $p_0 = \mathcal{G}[4K](g)$, wraps it as the initial chunk $c_0 = CAC(p_0, b[0:8])$, and uploads it to Swarm. If and when D receives the dream chunk c back, it extracts the dream key as the chunk payload $k = Data(c)$.

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

(1) The parity data C' is retrievable using standard retrieval methods.
(2) The dream protocol is followed by all cooperating nodes.
(3) The contents of the batch buckets along the dream path all remain unchanged.

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

Uploader U had granted downloader D at address a access to content C through the dream reference $\langle r, b, g\rangle$. U revokes D's access by uploading extra chunks to the buckets batch b designated by the input chunk address ($Address(c_i)$ for $0 \le i \le n$). As a result, downloader D will no longer be able to retrieve content C. Because the output of the key update function has changed, the subsequent chunk changes and the dream path is diverted. Since we cannot know which nodes are colluding and malicious, the uploader needs to upload batch bucket changes to all neighbourhoods on the dream path to reliably revoke access.

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 within the dream path, nor of any details about the other neighbourhoods that are part of it, except for the immediate next one to which they are push-syncing their chunk.

The construction of the dream reference is safely *deniable* as long as no adversary gets hold of the actual encrypted chunk of C': if we consider our sensitive content C, and any uncontroversial chunk content A, then when creating C', the owner also encrypts $A' = A \veebar k$ and uploads it. When asked about k, producing A' makes the denial of other content, including C', more plausible. For a scenario where it cannot be assumed that no adversary can get hold of C', both C

and A can be asymmetrically encrypted with a public key of the grantee before being encrypted with k, to prevent patterns showing in $A' \veebar C'$ that can help breaking the symmetric encryption and thus prevent successful deletion.[12]

As Swarm is a decentralised network, individual participants cannot (and should not) be trusted to play by the rules. The built-in incentives are what encourage cooperative behaviour and help maintain a Nash equilibrium. To avoid the situation that a single corrupt node in a neighbourhood could keep deleted chunks accessible, the mechanism for deletion, i.e. the strategic update to the relevant bucket, should be applied to every neighbourhood of a dream path. A sufficient condition for content deletion is then that in *at least one neighbourhood* all nodes are honest, i.e. will not store and serve information that they should not.[13]

Alternatively, 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 of what was 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 to the scenario where an adversary makes deletion redundant by keeping all downloaded data, which is a case not commonly interpreted as defeating deletion.

We now turn to the discussion of how to calibrate the step count in relation to the security model using the notion of a *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 safely revoked, the owner uploads new chunks in each neighbourhood along the dream path. In practice, the malicious node operators would not be able to tell which chunks are strategic updates to facilitate deletion. But we will assume for the sake of the argument 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. If every neighbourhood on the path contains at least one malicious node, it is, in principle, possible for the dream protocol to return 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 path, 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.[14] 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

$$k^{-n} \leq 10^{-\sigma} \tag{22}$$

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

$$n \geq \frac{\sigma}{\kappa} \tag{23}$$

[12] An adversary possessing both A' and C' could calculate $A' \veebar C'$ to find patterns in the result.

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

[14] This is a necessary condition, but not a sufficient one: even a malicious neighbourhood could act honestly, if the route happens to pass through one of its honest nodes. This further, significant improvement of the odds that deletion succeeds is for simplicity ignored in the following.

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 desired success rate.[15]

6 Implementation Constraints

Dream can be implemented on the existing Swarm network with only minor additions and modifications to the DISC protocol, its proven data storage mechanism. Because the dream protocol is simple and strategically utilises the DISC primitives, the effort required to implement it on Swarm would be modest.

Beyond its technical capacity, the size, speed, proven availability and maturity of the Swarm network[16] also make it suitable to serve as the underlying layer for the dream protocol. At a size of almost 10,000 nodes and a global spread across four continents, the network is large and distributed enough to provide the maze of nodes needed to make dream key creation and retrieval secure, and re-creation after intended deletion practically impossible.

The reference used by the grantees to retrieve content is more complex then Swarm's standard reference. Analogous to the way encryption is handled, the length of the reference allows to identify the referenced content as deletable.

7 Conclusion

Based on the concept of revocable access, this paper presents a technically novel yet practical mechanism to achieve deletable content in the immutable context of peer-to-peer storage. By formalising 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 DISC storage model and relies on simple yet robust protocol interactions among peer nodes. Through the introduction of a distributed key and a network-driven computation model, dream enables users to selectively revoke access to content without undermining the foundational principles of decentralisation, immutability, and censorship resistance. The result is a useful balance between data permanence and sovereign control, which offers a much-needed, viable approach to privacy-conscious publishing in the Web3 era.

The ability to delete data from immutable storage marks an essential contribution towards maturity for decentralised storage technology. It adds a feature that centralised, mutable systems so far seemed to have as a permanent advantage over the alternatives. Once established as a standard mode of storage, this can reduce regulatory friction and empower users. It could enable a new class of applications to utilise decentralised storage, which depend on compliance to be able to operate; and it caters to privacy-conscious users who wish stronger control over the data they share. While DREAM realises a subset of the 'right to be forgotten.' The fact that Swarm resists censorship while enabling deletion points to a promising middle ground as a potential direction for regulators of commercial data hubs.

The way that DREAM uses Swarm's network topology to 'store' a key–and the fact that a simple upload can change the conditions to effectively revoke access—may serve as an inspiration to rethink other seemingly intractable problems of web3.

Acknowledgments

The authors would like to thank György Barabás, Andrea Robert for help with text and formatting, Daniel Nickless for figure 1, and members of the Swarm Research Division, especially Daniel A. Nagy for their ideas leading to this paper.

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

[16] See the Swarm dashboard at https://network.ethswarm.org/ and the monitoring tool called Swarm Scan at https://swarmscan.io/

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] Petar Maymounkov and David Mazieres. 2002. Kademlia: A peer-to-peer information system based on the xor metric. In *Peer-to-Peer Systems*. Springer, 53–65.

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

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

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

[11] 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

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

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

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

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