

A New Standard in Data Security

As more and more data migrates to the cloud, alternatives to big-tech storage players are emerging with features that meet increasing security and privacy demands in new, innovative ways. How? By incorporating a decentralized architecture.

Decentralized architectures offer inherent data integrity benefits including wide distribution across geographies for fantastic disaster resilience, as well as resistance to ransomware and bitrot. An edge-based security model combined with delegated authorization provides flexible and secure access management capabilities—meaning only you and those you designate have access to your data.

The world's first cloud storage service powered through decentralization, Storj DCS (Decentralized Cloud Storage), delivers all of these inherent advantages plus a multi-layered security approach that includes pushing end-to-end encryption client-side, adding macaroon-based API keys, and providing simple, yet granular control. These features enable developers to reduce the threat surface on critical data to almost zero.

The Power Behind Storj DCS Secure Storage

End-to-End Encryption	Delegated Authorization	Simple Developer Tools
<p>Storj DCS offers end-to-end encryption and distributes erasure-coded pieces on geographically diverse Storj Nodes.</p> <p>Users control the automatically generated encryption keys—making it simple and secure.</p>	<p>Storj DCS pushes access management to the client and uses Macaroon-based API keys for an added layer of security.</p> <p>Pushing access management to the edge eliminates the need for access control lists, increasing privacy.</p>	<p>Storj DCS provides easy-to-use developer tools to manage both encryption and access for file sharing.</p> <p>Developers gain unprecedented control to maintain privacy while securely sharing data.</p>

A Multilayered Approach for Secure Encryption

Storj DCS utilizes multiple encryption technologies that work in concert to ensure your data is secure—giving you absolute control over how your data is accessed.

End-to-End Encryption	Path-Based Encryption	Content & Metadata Encryption	Distributed Data Storage	Encryption Key Ownership
When using Storj DCS via CLI, library, or local gateway, every object is End-to-end encrypted to ensure that only users have access to their data.	Each path is encrypted separately in a hierarchical and deterministic way using the root encryption key.	All file segments and metadata are separately encrypted so only you know what is stored on Storj DCS.	The encrypted, erasure-coded file segments are spread over geographically diverse Storj Nodes.	Your encryption keys and data are yours. Storj DCS never has access to them.

Storj DCS Encryption Methodology

When using Storj DCS via CLI, library, or local gateway, all data stored is end-to-end encrypted on the client-side. What this means is users control all encryption keys and the result is an extremely private and secure data store. Both the objects and the associated metadata are encrypted using randomized, salted, path-based encryption keys. The randomized keys are then encrypted with keys derived from the user's encryption passphrase. Neither Storj nor any Storj Nodes have access keys, data, or metadata.

Client-Side Encryption

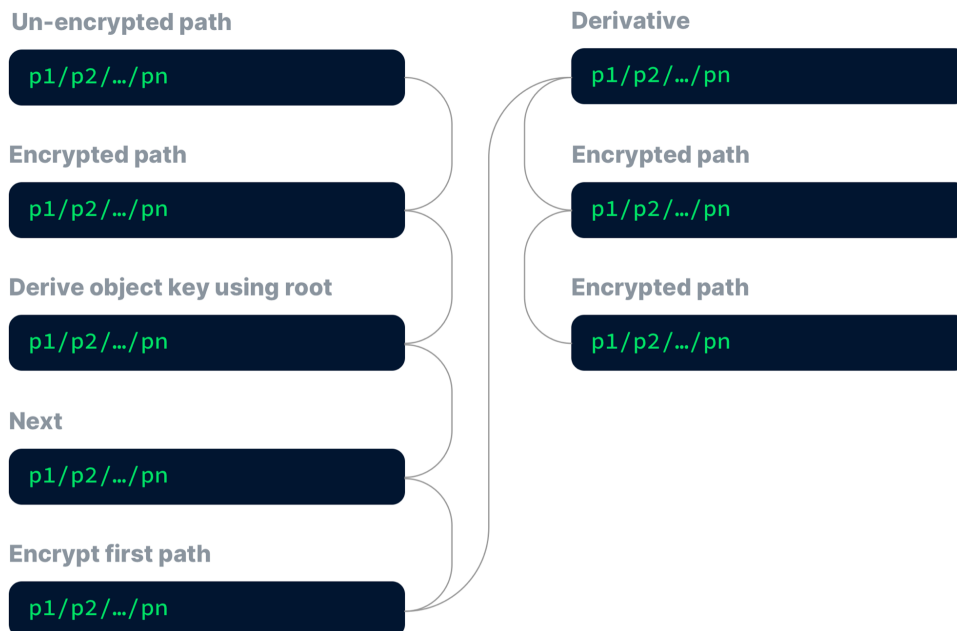
Every object stored on the platform is first split into 64 MB segments, then encrypted using one of two included encryption schemes, the default AES-GCM 256 CTR, or Secretbox. The encryption scheme is designed to be pluggable, meaning developers may also integrate custom encryption schemes if requirements dictate it.

This encryption is designed to avoid using the same keys for content encryption of different files and different segments of the same file. This is advantageous because it makes sharing of encrypted files more secure and it doesn't put other segments or files at risk if one of them is compromised.

Path-Based Encryption

Paths are encrypted in a hierarchical and deterministic way using the root encryption key. Each path component is encrypted separately based on information derived from previous path components.

Consider an unencrypted path p made up of path elements $p_1/p_2/.../p_n$. The end goal is to generate an encrypted path e , which is made up of elements $e_1/e_2/.../e_n$. We achieve this using the process shown.



The order of listed items is determined by the paths stored on the Satellite. Listed items will always be returned in an order based on their encrypted path names, but won't be ordered alphabetically when the paths are decrypted. This method of path encryption allows users to share content under their path with another user without revealing anything at a higher level.

Content and Metadata Encryption

When a user uploads a file, it's read one segment at a time on the client-side. Before each segment is split up, erasure encoded, and stored on remote Storj Nodes, a random content-encryption key is generated. A starting nonce equal to the segment number is created and used, along with the random key, to encrypt the segment data.

Next, we generate the derived key, dk , which we define with $sn+1 = \text{HMAC}(sn, \text{"content"})$, where $dk = K(sn+1)$ and sn is the last secret generated from the file path using the technique detailed above. We add one more dimension of key derivation for content encryption to ensure a user can't derive the access key to an unshared file that has the same prefix.

ENCRYPTION HIERARCHY

```
[  
  Root\path1  
  Root\path1\path2  
  Root\path1\path2\path3  
]
```

Each segment has metadata associated with it on the Satellite. Segment metadata includes the random key used to encrypt a segment's content. We encrypt the random key with the derived key (dk) and a randomly generated nonce. The nonce is stored along with the encrypted content key in the segment metadata. This way, we use a different random encryption key for each segment, but anyone with access to the derived key can decrypt those keys.

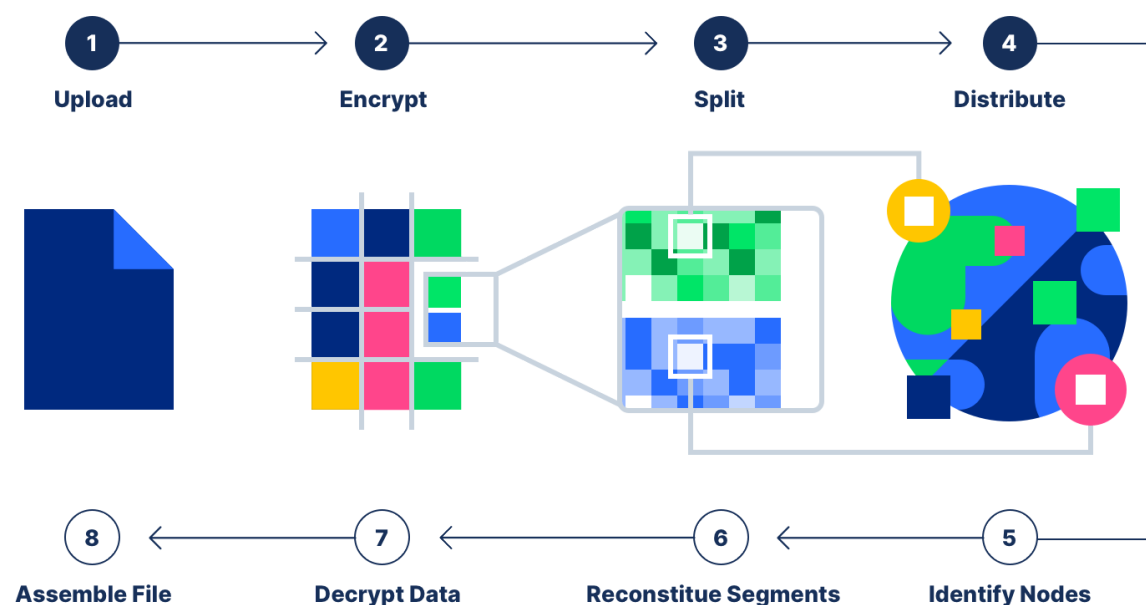
The encryption algorithm used for content and metadata is AES-GCM, but being open source, other algorithms could be used. AES-GCM is an authenticated encryption algorithm, which means if any encrypted data is tampered with, the client downloading

the data will know about it once the data is decrypted. The layers of content encryption mean only you know what you've stored on the platform.

Distributed Data Storage

Privacy and security are further enhanced as the encrypted, erasure-coded pieces of each segment are spread over diverse Storj Nodes around the world. Any effort to compromise a file requires significant coordinated effort beyond the effort to crack a single encryption key.

Take for example a 1GB file stored on Storj DCS. The file is split into 16 different segments, with 16 different randomized, salted path-based segment encryption keys, each segment divided into 80 or more pieces, with a total of 1,280 pieces stored on 1,280 different Storj Nodes.



If an individual Storj Node Operator went rogue, it could at most gain access to a single encrypted, erasure-coded piece of a single file; this piece is only one out of twenty-nine pieces of one segment out of sixteen, that comprise the file. No usable information can be derived and if the piece is deleted, Storj DCS detects this and repairs as necessary.

Encryption Key Ownership

Your encryption keys are your data. Unless you deliberately share them with us, Storj DCS never has access to your encryption keys. We provide all the tools to ensure your data security and privacy are protected, along with the ability to share that data when, how, and with whom you allow. If you lose your encryption keys, however, then you've lost access to your data. Always make sure you've backed up your encryption key and stored it somewhere safe.

Sophisticated Access Management with Macaroon-Based API Keys

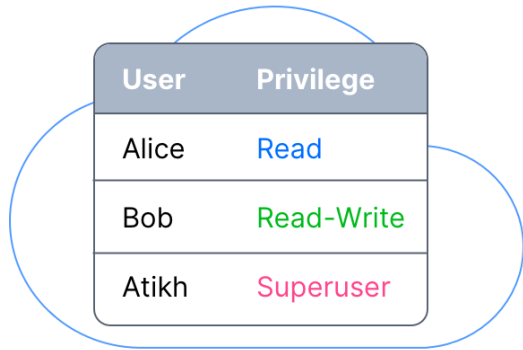
While encryption is very important to ensure privacy and security for data storage, control over when and how data is shared and accessed is equally important. Storj DCS provides macaroon-based API keys for managing access to objects.

Hierarchically Derived (HD) API Keys

Similar to hierarchically derived (HD) encryption keys, HD API keys are derived from a parent API key. Unlike the HD encryption keys where the hierarchy is derived from the path prefix structure of the object storage hierarchy, the hierarchy of API keys is derived from the structure and relationship of access restrictions.

HD API keys embed the logic for the access it allows and can be restricted, simply by embedding the path restrictions and any additional restrictions within the string that represents the macaroon. Unlike a typical API key, a macaroon isn't a random string of bytes, but rather an envelope with access logic encoded in it.

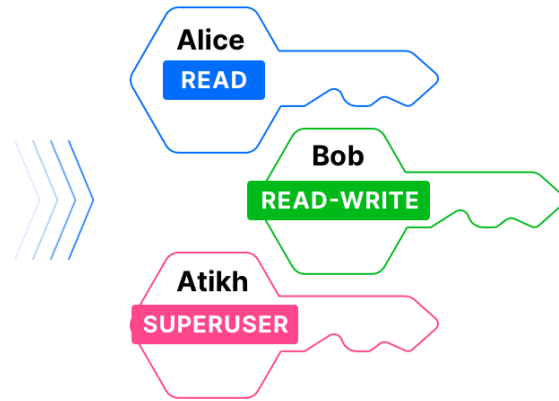
Centralized ACL



A table with two columns: 'User' and 'Privilege'. It lists three users: Alice with 'Read' privilege, Bob with 'Read-Write' privilege, and Atikh with 'Superuser' privilege. The table is enclosed in a blue oval.

User	Privilege
Alice	Read
Bob	Read-Write
Atikh	Superuser

Decentralized Capabilities



Root API

```
[  
  Root API [Caveat \path1\path2\path3]  
  Root API [Caveat \path1\path2\path3] + [Caveat IP Range]  
  Root API [Caveat \path1\path2\path3] + [Caveat IP Range] + [Caveat Date/Time Limit]  
]
```

Macaroon-based API Key



Benefits of Hierarchically Derived (HD) API Keys

#1: No Centralized Access Control Lists - Access controls are generated client-side then verified and interpreted server-side, eliminating the need for centralized access control lists.

#2: More Secure Identity & Access Management (IAM) - By tying access to HD API keys, rather than a centralized control system, capability-based models push security to the edge, creating a more secure IAM system.

#3: Full Control Over Data Management - Key-based ownership of object data enables users to intuitively control their data as a first principle, and then delegate it as they see fit.

#4: No Risk of Data Loss or Extortion - The decentralized cloud eliminates the increasingly apparent risk of data loss/extortion due to holding data on one single centralized provider.

Supported Caveats for HD API Keys

API keys delegate access to a bearer token that can only be used in specific circumstances through HMAC chained 'caveats' (i.e. restrictions on IP, time-server parameters, and third-party auth discharges). These caveats can be extended and chained, but not overwritten.

The specific caveats supported by Storj DCS are as follows:

- **Specific Operations:** Caveats can restrict whether an API Key can perform any of the following operations: Read, Write, Delete, List.
- **Bucket:** Caveats can restrict whether an API Key can perform operations on one or more Buckets.
- **Path and Path Prefix:** Caveats can restrict whether an API Key can perform operations on objects within a specific path in the object hierarchy.
- **Time Window:** Caveats can restrict when an API Key can perform operations on objects stored on the platform, either before or after a specified date/time or between to dates/times.

From a developer standpoint, HD API keys make it very easy to write code that granularly defines security privileges. Once baked, the rules within the HD API key can't be changed, without reissuing the key itself.

Best-In-Class Developer Controls

As a developer platform, client-side encryption, erasure coding, and distributed storage ensure your application data is extremely secure. We don't stop there, however, we also provide simple to use access management controls so your users can manage access to their data.

A Simple Command for Access Management: The "Access Grant"

Access management on Storj DCS requires coordination of the two parallel constructs—encryption and authorization. Both of these constructs work together to provide a client-side access management framework that's secure and private, as well as extremely flexible for application developers.

When sharing access to objects, Storj DCS requires sending encryption and authorization information about an object from one client to another. The information is sent in a construct called an Access Grant. An Access Grant is a security envelope that contains a path, a restricted HD API key, and an HD encryption key—everything an application needs to locate an object on the network, access that object, and decrypt it.

To make the implementation of these constructs as easy as possible for developers, the Storj DCS developer tools abstract the complexity of encoding objects for access management and encryption/decryption. A simple share command encapsulates both an encryption key and a macaroon into an Access Grant in the format of an encoded string that can be easily imported into an uplink client. Imported Access Grants are managed client-side and may be leveraged in applications via the uplink client library.

Designed for Efficient File Sharing

While the API key is intended to be passed from an Uplink Client to a Satellite, the encryption key is designed to be passed peer-to-peer, but never to the Satellite. For efficiency and ease of use, the file-sharing functions of the Uplink Client constructs the

Access Grant that contains both the API key and the encryption key that's passed peer-to-peer. An application that receives an Access Grant and imports it then passes the API key to the appropriate Satellite to obtain access to the object, and then, as the object is downloaded, it's decrypted client-side using the HD encryption key.

Access Management at the Edge Delivers Differentiated Value

The edge-based security model of Storj DCS provides easy tools for building applications that are more private, more secure, and less susceptible to the range of common attacks. Unlike other cloud storage providers, Storj DCS security features are enabled by default—at no extra cost.

Storj DCS Security Features Deliver:

- **Significantly Reduced Risk of Common Attacks** - Storj does not store your keys, which means that common attacks that depend on breaching a central repository of decryption keys are impossible.. The Storj DCS security model eliminates whole categories of typical application attack vectors.
- **Reduced or Eliminated Typical Threat Surfaces** - By separating trust boundaries and distributing access management and storage functions, a significant percentage of the typical application threat surfaces is eliminated or made orders of magnitude more complex to attack.
- **Enhanced Data Privacy** - With access managed peer-to-peer, the platform can separate responsibilities for creating bearer tokens for access management from encryption to use the data. Separation of these concerns enables decoupling storage, access management, and use of data, ensuring greater privacy with greater transparency.
- **Delegated Authorization to the Edge** - Authorization delegation is decentralized and managed at the edge but derived based on a common, transparent trust framework. This means that access tokens generated at the edge can be efficiently interpreted centrally but without access to the underlying encrypted data. This maximizes privacy and security at scale.
- **No Added Cost for Security Features** - Other cloud storage providers have a separate product and associated cost for security features. For instance, the AWS Detective solution. Yet even with the additional cost, the security capabilities do not match up to the power and control of the Storj DCS platform—which includes these capabilities out-of-the-box.

Storj DCS is Purpose-Built for Distributed Data

Distributed data storage architecture combined with edge-based encryption and access management stores your data as if it were encrypted and stored on an encrypted beach. The combination of client-side HD Encryption keys and HD API keys in an easy-to-use platform enables application developers to leverage the capability-based security model to build applications that provide superior privacy and security.

Glossary of Terms

- **Access Grant:** An encoded string that contains both a path key and an HD API key for the purpose of sharing access to objects stored on the platform.
- **AES-GCM:** An authenticated encryption algorithm to make use of the Advanced Encryption Standard and uses the Galois/Counter mode for encrypting blocks.
- **API Key:** A string generated for a project to authorize access management to data on the platform. Our API Keys are macaroons.
- **Caveat:** An access restriction encoded into an HD API key, generated client-side, and is interpreted by a Satellite.
- **Derived Key:** A key derived from the path key for the lowest level path element. The derived key is used to encrypt the random key before it is stored in a segment's metadata.
- **HD API Key:** A string generated from an API key (hierarchically derived) with one or more caveats to authorize restricted access to data on the platform.
- **HMAC:** Hash-based message authentication code. We generate HMACs using path elements and encryption keys in order to derive new keys for lower levels of the path. Using hashes makes it easy to generate keys from higher levels without making it possible to generate higher-level keys from lower-level ones.
- **Macaroon:** Authorization credentials provide flexible support for controlled sharing in decentralized, distributed systems. Unlike a typical API key, a macaroon isn't a random string of bytes, but rather an envelope with access logic encoded in it.
- **Object Encryption Key:** A key derived from the root secret and the file path. There's a different path key for every element in the path, and a path key is used to derive new path keys for lower-level path items.
- **Object Key or Path:** The representation of a file's "location." Paths are essentially an arbitrary number of strings delimited by slashes (e.g. `this/is/a/file.txt`). On the Storj network, the Satellite uses paths to keep track of file metadata as well as pointers to storage nodes that possess encrypted file content.
- **Random Key:** A randomly generated key used to encrypt segment content and metadata.
- **Root Secret:** The private client-side encryption key defined in the client configuration used to derive keys for encrypting and decrypting data stored on the platform.

- **Secretbox:** An authenticated encryption algorithm from the NaCl library that combines the Salsa20 encryption cipher and Poly1305 message authentication code.
- **Segment:** The largest subdivision of a file. All the segments of a file are usually the same size. In most cases, the last segment will be smaller than the rest.

Start Building on the Decentralized Cloud

Decentralization is already here, and it's only going to get bigger, better, and more mainstream as people discover the benefits of a decentralized model. Head over to www.storj.io and see how the unparalleled privacy and security features of Storj DCS can start benefiting you, your project, and your organization today.



Start building on the decentralized cloud.

www.storj.io



© 2021 Storj Inc.