Nextcloud

End-to-end encryption design

Version: September 20, 2017

nextcloud.com

Nextcloud

Contents

Introduction

With the announcement of the Nextcloud end-to-end encryption tech-preview, we'd like to invite you to scrutinize our source code and cryptographic approach.

Please note that end-to-end encryption feature is a work-in-progress and this document may describe functionalities or approaches not yet implemented in our testing releases. This document is meant as authoritative implementation guideline for our clients.

For the sake of having smaller and incremental steps towards the final implementation we're going to continuously release updated builds of our clients.

We are looking forward to your input to refine our approach towards client side encryption. In addition, we will also make sure to validate our approach on-time by external cryptographic experts.

Requirements

The end-to-end encryption has to fulfill the following business and technical criteria.

Security properties

The following security properties have to be fulfilled:

- Access to ciphertext must not leak directory structure nor file names or content.
  - Leaking the number of files in an encrypted folders is an accepted risk.
- Public keys of users must be auditable
- Once a user has been removed from an encrypted folder they should have no relevant key material to encrypt files updated or created in the future
- Encrypted folders must use an encryption scheme fulfilling the following criteria:
  - Confidentiality
    - No one, except the legitimate recipients, must have access to the encrypted documents.
  - Integrity
    - Even with writable access to the ciphertext one should not be able to tamper with the data. In case an encrypted referenced file is deleted from the file system but still found in the metadata a warning should be displayed to the user.
  - Authenticity
    - Authenticity of the files has to be guaranteed.

Usage of widely available and tested libraries for crypto primitives

We believe that for security-sensitive functionalities relying on existing and proven libraries is an essential requirement. Thus we require that:

- The used library for cryptographic primitives must be in use widely.
- The used library for cryptographic primitives has undergone successful security audits.

Also due to our wide range of supported systems, the library must be available for the following of our supported environments:

- iOS 9+
- Android 6.0+
- Mac OS X 10.9+
- Windows 7+
- commonly used Linux distributions
- PHP 7.0+

Note: While we don't have any current plans to add support for potential server-side decryption we want to keep this possibility open for the future.

Sharing functionality

Existing client-side encryption solutions often prevent the sharing of encrypted files, the Nextcloud end-to-end encryption must offer support for the following sharing scenarios:

- Sharing encrypted folders with other users

The following sharing scenarios are considered out of-scope:

- Sharing single files or folders from an encrypted folder
- Sharing encrypted folders with whole groups

Optional central data recovery

While End-to-End encryption is meant to prevent access to data for other parties the reality is: People may lose their encryption keys. While in an home user environment this may be acceptable, in an enterprise this can have grave implications. Thus an optional central data recovery has to be available offering the following capabilities:

- Central recovery key per instance can be generated
- Central recovery key must not be stored on the instance and can be safely exported (e.g. to be stored in a physical vault)
- All data will also be encrypted to the central recovery key when enabled
- Users must be prominently warned in the UI of their clients if a central data recovery key is enabled
- When a central data recovery key is enabled the existing end-to-end encrypted folders must not be affected

Simple multi-device management

Access to encrypted data should easily be possible from any device the end-user owns, this includes all mobile devices as well as desktop devices.

Thus:

- Sharing keys between existing devices must be frictionless
- Newly added devices should have access to all previously encrypted data

Simple authenticated key exchange

Key exchange is a key problem of any cryptographic system, on one hand one wants to ensure that the key of the participating parties is authentic. On the other hand, manual comparisons of fingerprints are cumbersome and rarely something that regular users can be bothered to do.

A secure and yet simple system has to implement the following properties:

- Key exchange between parties should be frictionless
- Exchanged keys should be auditable

Support for HSM

To fulfill enterprise security requirements it should be possible that key material is generated by a hardware security module. Thus offering strong authentication, tampering resistance and a complete audit trail.

Versioning

The protocol has to support versioning in case of future changes in the metadata or cryptographic handling.

Accepted feature loss

Since the data is not accessible to the server and to simplify the implementation a loss of the following features is acceptable:

- Server-Side trash bin
- Server-Side versioning
- Server-Side search
- Server-Side previews
- Access to folders via web interface
- Sharing to groups
- Sharing at the level of individual files

Technical implementation

The encryption is based upon an asymmetric cryptographic system. Every user has exactly one private and public key pair. The following steps will walk through the current technical implementation of the encryption.

Adding an end-to-end encrypted device

As a first step a device has to be added to an account, a device can be anything able to run one of our supported clients.

Depending on whether an end-to-end encrypted device already has been added to an account, the device will have to create new key material or use existing key material. To check whether a certificate has already been issued or not the `/ocs/v2.php/apps/end_to_end_encryption/api/v1/public-key` endpoint should be used.

In addition, the client has to download the server's public certificate from `/ocs/v2.php/apps/end_to_end_encryption/api/v1/server-key` and use this to verify the certificate chain in all future operations.

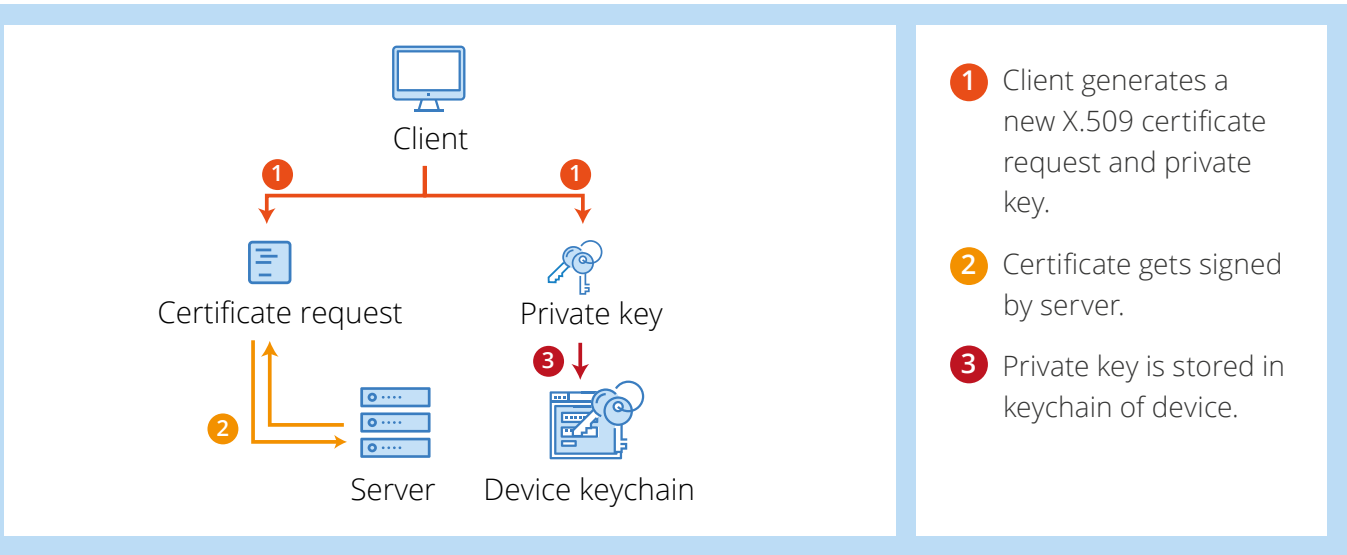
Initial device

When a device is initially added to an account the device has to generate all relevant key material for the user account and provision those on the server.

First, the client has to generate the relevant key material:

1. Client has to generate a new X.509 certificate request and private key.
    1. CN of X.509 certificate must be set to the currently logged-in User ID
2. Client uploads the X.509 certificate request to the server by sending the certificate request URL encoded as parameter `csr` to `/ocs/v2.php/apps/end_to_end_encryption/api/v1/public-key`.
3. Server issues a certificate if the CN matches the current user ID.
4. Server returns the issued certificate.
5. Client stores the private and the public key in the keychain of the device.

Graphic: How to create and sync identity (Part 1)

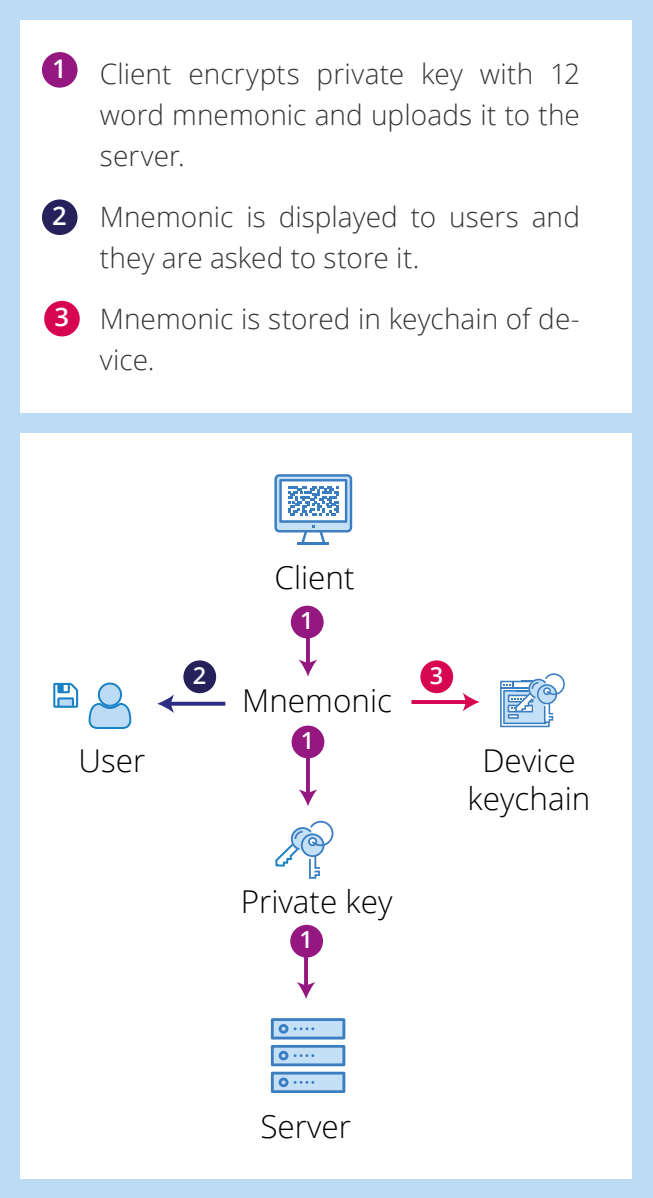


In a second step, the private key will be stored encrypted on the server to simplify the addition of further devices:

1. Client generates a 12 word long mnemonic from the English BIP-0039 word list. The word list contains 2048 words, thus resulting in 2048^12 possible key combinations.
2. Client encrypts the private key using AES/GCM/NoPadding as cipher (128 bit key size) and uses PBKDF2WithHmacSHA1 as key derivation, as password the in step 1 generated mnemonic is used.
3. Client uploads the encrypted X.509 private key to the server by sending the encrypted private key URL encoded as parameter `privateKey` to `/ocs/v2.php/apps/end_to_end_encryption/api/v1/private-key`.
4. The mnemonic is displayed to the user and the user is asked to store a copy in a secure place.
5. The mnemonic is stored in the keychain of the device.

In case a user loses their device they can easily recover by using the mnemonic passphrase. The mnemonic passphrase can also be shown in the client settings in case the user forgets their mnemonic. Displaying the mnemonic requires the user to enter their PIN/fingerprint again on mobile devices.

Graphic: How to create and sync identity (Part 2)

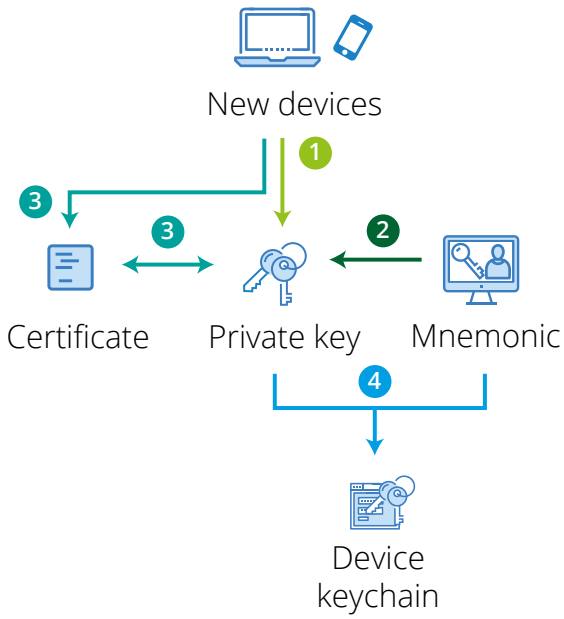


Further devices

In case a certificate exists already for the user the client has to download the existing private key. This is done the following way:

1. Client downloads private key from the `/ocs/v2.php/apps/end_to_end_encryption/api/v1/private-key` endpoint.
2. Client asks the user for the mnemonic and decrypts the private key using AES/GCM/NoPadding as cipher (128 bit key size) and PBKDF2WithHmacSHA1 as key derivation.
3. Client checks if private key belongs to previously downloaded public certificate
4. Client stores the private key in the keychain of the device.

Graphic: How to add further devices



1. New devices download the encrypted private key.
2. Private key gets decrypted with the 12 word mnemonic from the user.
3. New devices check if private key belongs to certificate.
4. Private key and mnemonic are stored in the keychain of the device.

Creating an end-to-end encrypted folder

To create an end-to-end encrypted folders multiple steps have to be performed. First of all, data access to such folders happens via our regular WebDAV API available at `/remote.php/dav/$userId/files`.

Mark folder as end-to-end encrypted

After creating a folder via WebDAV the folder has to be flagged as end-to-end encrypted, this can be performed by sending a PUT request to `/ocs/v2.php/apps/end_to_end_encryption/api/v1/encrypted/<file-id>` whereas `<file-id>` has to be the file ID indicated by our WebDAV API.

Once this flag has been set the folder will not be accessible anymore via web and also not displayed to regular DAV clients. Only empty folders can be marked as end-to-end encrypted.

Create metadata file

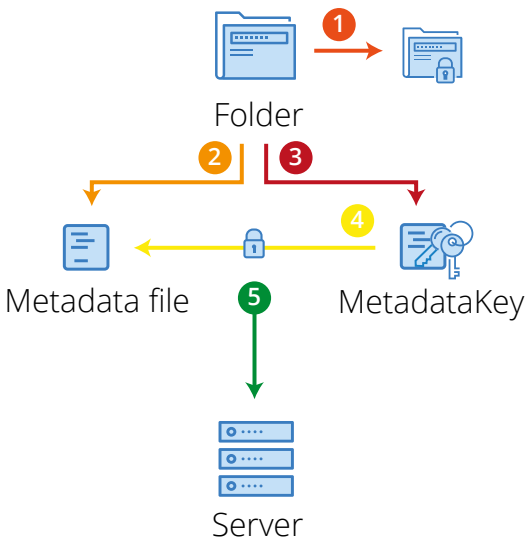
Every folder contains a metadata file containing the following information:

- Metadata of files (filename, mimetype, ...)
- Access list to the folder
- Key material for files in the folder

The metadata is a JSON document with the following structure. The `metadata->metadataKeys` elements are encrypted to the recipients public keys and the values are used to encrypt the single file metadata elements.

In case the central data recovery key is enabled the metadata will also be encrypted towards the servers central data recovery key. Clients must show a prominent warning to the users for such scenarios.

Graphic: How to encrypt a folder



1. Create folder and mark to server as encrypted.
2. Generate metadata file.
3. Generate metadataKey, encrypted to all public keys that have access to the folder.
4. Use metadataKey to encrypt all values in metadata file.
5. Store encrypted meta-data on server.

The only unencrypted elements in the JSON document is the version of the metadata file. The other informations are all encrypted either based on the public key or the actual metadata keys. The encrypted JSON array elements should just be encrypted as simple string element. This means that “foo => [bar, foo]” should become “foo => “ciphertext” and the clients are responsible for decoding this ciphertext in a proper array again

The metadata has to be created by sending a POST request to `/ocs/v2.php/apps/end_to_end_encryption/api/v1/meta-data/<file-id>`, whereas <file-id> has to be the file ID indicated by our WebDAV API. As POST parameter `metaData` with the encrypted metadata has to be used.

```
{
        // Metadata about the share
        "metadata": {
                // Following, each metadata key is encrypted to all public keys that
                have access to the share. It is generated by client on first upload.
                // A blob of keys useful for key rotation. If a recipient has been
                removed of a share a new metadata key will be generated and the client
                always uses the newest one.
                // Encrypted files refer to which metadata key to use. In case of
                updating a file the client should update it with the new metadata
                encryption key.
                // Encryption algorithm: RSA/ECB/OAEPWithSHA-256AndMGF1Padding,
                encrypted via private/public key (asymmetric)
                "metadataKeys": {
                        "0": "OLDESTMETADATAKEY",
                        "2": "…",
                        "3": "NEWESTMETADATAKEY"
                },
                // The following blob contains the reference to all keys that have
                access to the share.
                // Encrypted payload to the currently used metadata key
                // Encryption algorithm: AES/GCM/NoPadding (128 bit key size)  with
                metadata key from above (symmetric)
                "sharing": {
                        // Name of recipients as well as public keys of the recipients
                        "recipient": {
```

```
                                "recipient1@example.com": "PUBLIC KEY",
                                "recipient2@example.com": "PUBLIC KEY"
                        },
                },
                // The version of the metadata file
                "version": 1
        },
        // A JSON blob referencing all files
        "files": {
                // Following blob refers to the encrypted file "ia7OEEEyXMoRa1QWQk8r" on the filesystem
                "ia7OEEEyXMoRa1QWQk8r": {
                        // Encrypted payload to the currently used metadata key
                        // Encryption algorithm: AES/GCM/NoPadding (128 bit key size) with metadata key from above (symmetric)
                        "encrypted": {
                                // Encryption key of the file
                                "key": "jtboLmgGR1OQf2uneqCVHpklQLlIwWL5TXAQ0keK",
                                // Unencrypted file name
                                "filename": "/foo/test.txt",
                                // Mimetype, if unknown use "application/octet-stream"
                                "mimetype": "plain/text",
                                // Which encryption method version was used? For updating in the future.
                                "version": 1
                        },
                        // Initialization vector
                        "initializationVector": "+mHu52HyZq+pAAIN",
                        // Authentication tag of the file
                        "authenticationTag": "GCM authentication tag",

                        // Which metadata key to use
                        "metadataKey": 1
                }
        }
}
```

Update metadata file

To keep the metadata and the file in sync locking is required. The client needs to lock the encrypted folder. If the lock operation succeeded the file the server will return a successful response together with a token in the response body. In case of a lost connection the client can restart the operation later with another "lock" request, in this case the client should send the token with the new lock call. This enables the server to decide if the client is allowed to retry the upload.

After locking was successful, the client will upload the encrypted file and afterwards the metadata file. If both files are uploaded successful the client will finish the operation by sending a unlock request.

To lock the metadata a POST request to `/ocs/v2.php/apps/end_to_end_encryption/api/v1/lock/<file-id>` has to be sent. Whereas `<file-id>` has to be the file ID indicated by our WebDAV API. To add an existing lock token it can be sent as `token` parameter.

To update the metadata a PUT request to `/ocs/v2.php/apps/end_to_end_encryption/api/v1/meta-data/<file-id>` has to be sent. Whereas `<file-id>` has to be the file ID indicated by our WebDAV API. As parameters "token", which contains the current lock token, and "metadata", containing the encrypted metadata have to be sent.

To unlock the metadata a DELETE request to `/ocs/v2.php/apps/end_to_end_encryption/api/v1/lock/<file-id>` has to be sent. Whereas `<file-id>` has to be the file ID indicated by our WebDAV API. The previously received lock token has to be sent as `token` parameter.

Uploading a file into an end-to-end encrypted folder

To upload a file in an end-to-end encrypted folder the client has to differentiate whether it is a new file or an existing file that gets updated.

The client can verify whether it is a new file or an existing one by downloading the metadata and checking if the "files" array contain the referenced file.

Uploading new files

In case a new file is uploaded the client has to do the following steps:

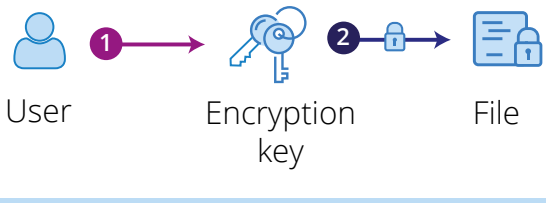
1. Generate a new 128-bit encryption key for the file and encrypt it using AES/GCM/NoPadding.
2. Generate a random identifier for the file (e.g. an UUID) and upload the encrypted file via WebDAV using the random identifier as file ID
3. Add new file to the files array in the metadata file
4. Update and lock the encrypted folder as described in "Update metadata file". The latest metadataKey should be used to encrypt the metadata.

Updating existing files

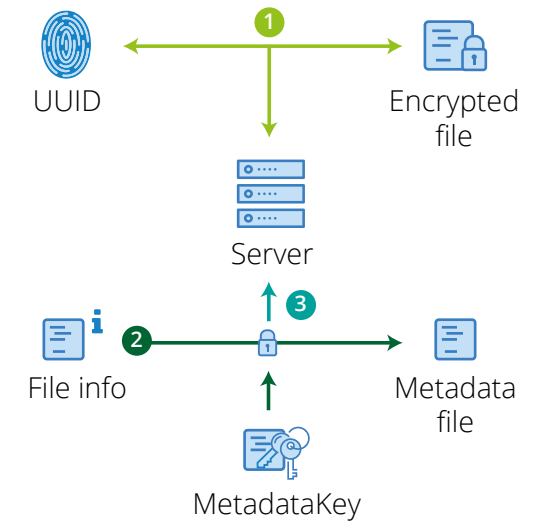
In case an existing file is updated the client has to do the following steps:

1. Generate a new 128-bit encryption key for the file and encrypt it using AES/GCM/NoPadding.
2. Lock the encrypted folder
3. Use the existing random identifier for the file and upload the encrypted file via WebDAV using the existing random identifier as file ID
4. Update the file in the files array of the metadata
5. Update and unlock the metadata file. The latest metadataKey should be used to encrypt the metadata.

Graphic: How to upload new files



1. Generate new 128bit encryption key for file.
2. Encrypt it using AES/GCM/NoPadding.



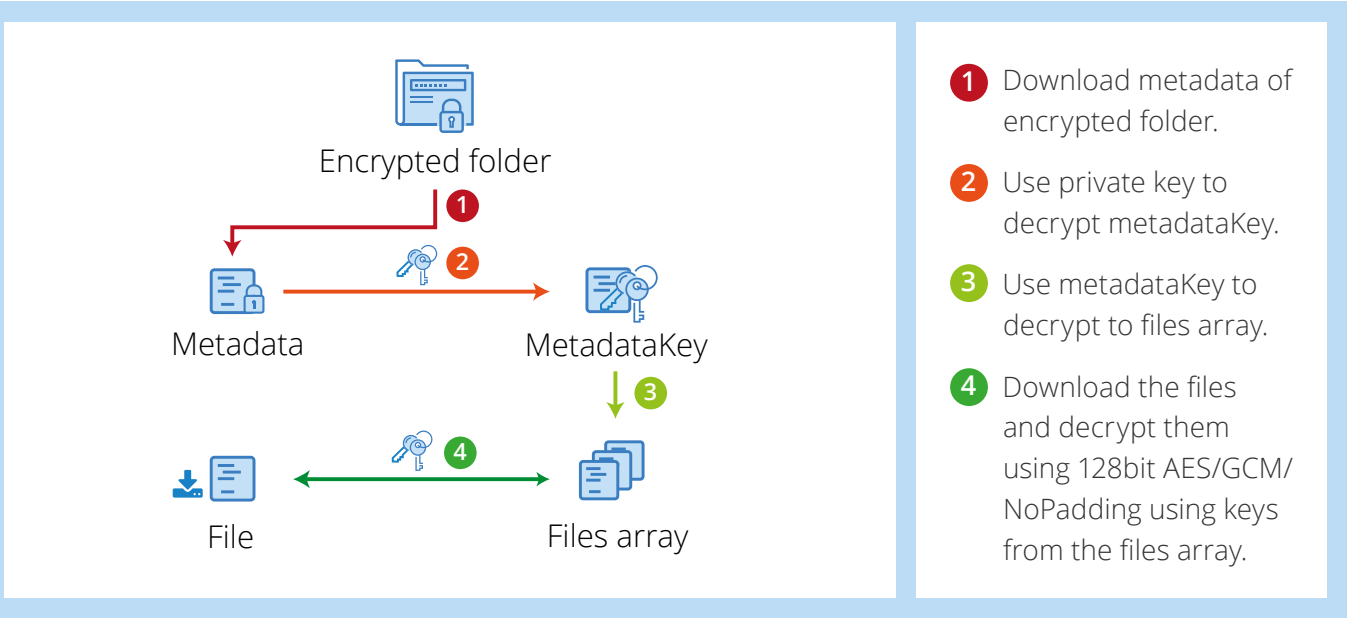
1. Generate random identifier (UUID) and upload encrypted file, using random identifier as file ID.
2. Add new file info to files array in metadata file, encrypted with metadataKey.
3. Update metadata on server.

Accessing encrypted files

To access encrypted files the client has to do the following steps:

1. Download actual metadata of encrypted folder
2. Loop over “files” array and decrypt the array with the newest metadataKey. Use metadataKey to decrypt to files array.
3. Download the referenced files using WebDAV and decrypt using AES/GCM/NoPadding (128bit) and using the referenced file keys in the file array.

Graphic: How to access encrypted files



In case a file is referenced in the metadata but cannot be found on the WebDAV file system the user should be warned about this. If the file exists locally but not on the file system the client should re-upload the file.

Sharing encrypted folders to other users

Key discovery of other users

As a PKI approach for encryption is used every certificate is issued by a central root authority. By default the Nextcloud server acts as a Root Authority and issues the certificates from the CSRs.

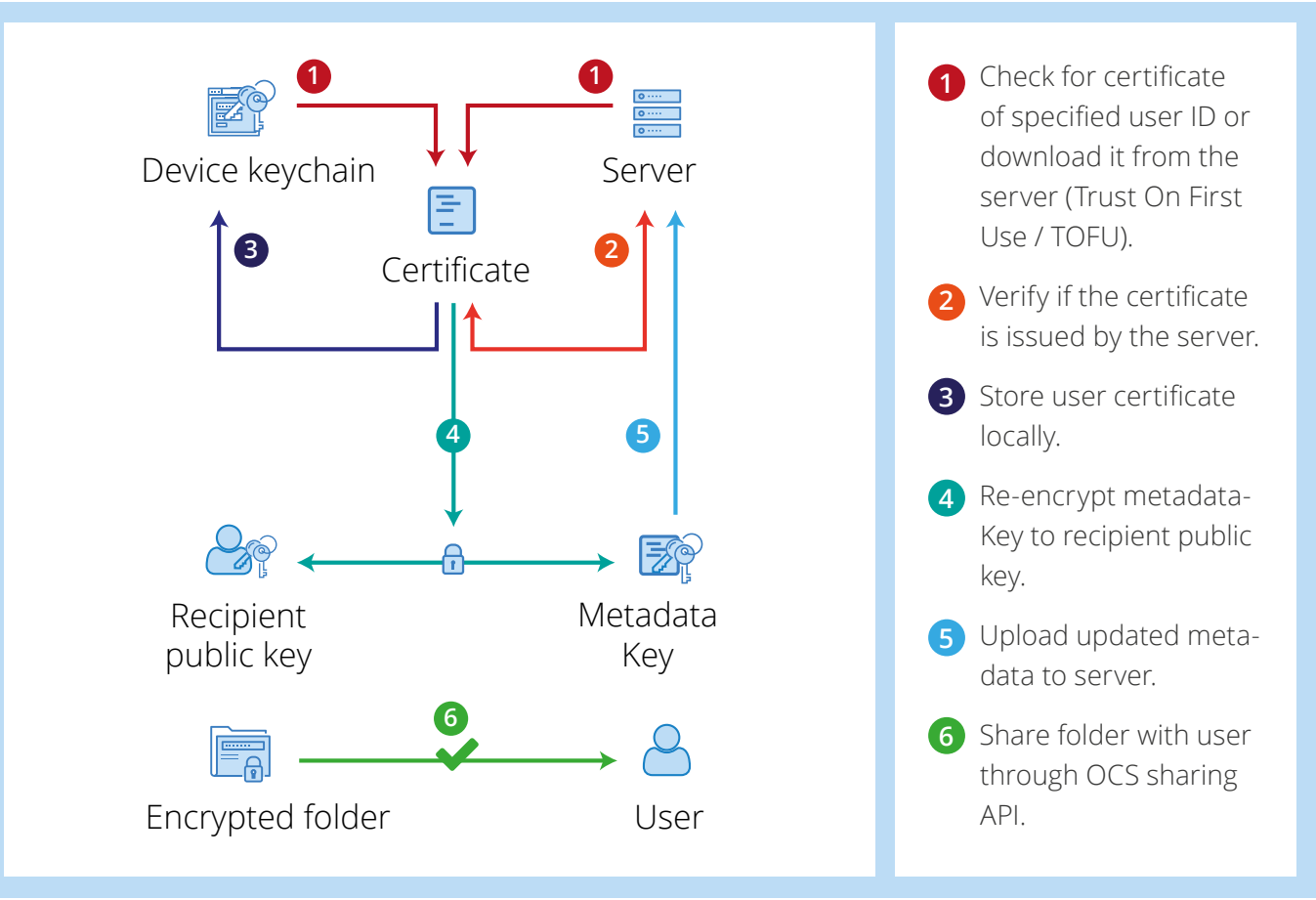
The clients do the following when trying to establish a trust relationship to another user:

1. Check if a certificate for the specified User ID is already downloaded (Trust On First Use (TOFU))
    1. If an certificate is available this one will be used
    2. If none is available the client will continue at 2.

2. Query the user certificates by sending GET request to the `/ocs/v2.php/apps/end_to_end_encryption/api/v1/public-key` endpoint and sending a JSON encoded users parameter containing the specified UIDs
3. Verify if the certificate is issued by the downloaded server public key.
    1. If yes: Use this one.
    2. If no: Show a warning that initiating an encrypted share isn't possible to the user.
4. Store the user certificate locally for next TOFU operations

Note: We're considering adding support for additional security measures such as Certificate Transparency Logs or HSM devices. Thus further reducing the risk of a hacked server.

Graphic: How to share an encrypted folder



Add someone to an end-to-end encrypted folder

To create a share the following actions have to be performed:

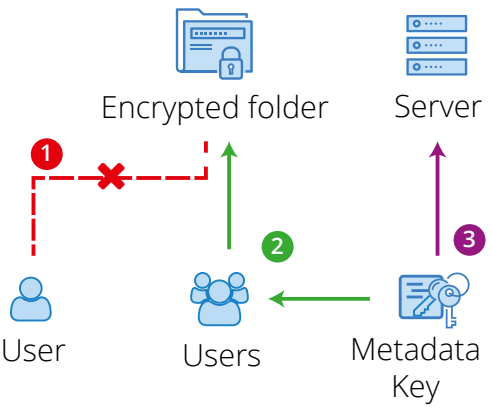
1. The file has to be shared via the OCS sharing API to the recipient
2. The metadataKeys must be encrypted to the recipient public key
3. The recipient is added to the "sharing" array

Remove someone from an existing share

To remove someone from an existing share the following actions have to be performed:

1. The file has to be unshared via the OCS sharing API to the recipient
2. A new metadataKey must be generated
3. The recipient is removed from the “sharing” array
4. The metadata-key array must be re-encrypted to everyone except the recipient

Graphic: How to remove someone from an existing share



1. Unshare encrypted folder with user through OCS sharing API.
2. Generate new metadataKey and encrypt to everyone who now has access.
3. Upload metadata to server.

Edgecases

Handling of complete key material loss

Right now a complete key material loss means that other users that already had a share with the user will not be able to share new encrypted folders since the protocol uses TOFU for initiating shares.

However, considering the fact that the user has a mnemonic passphrase to recover their key and any connected device (e.g. their smartphones) also has a way to recover the mnemonic we consider this an edge-case at the moment.

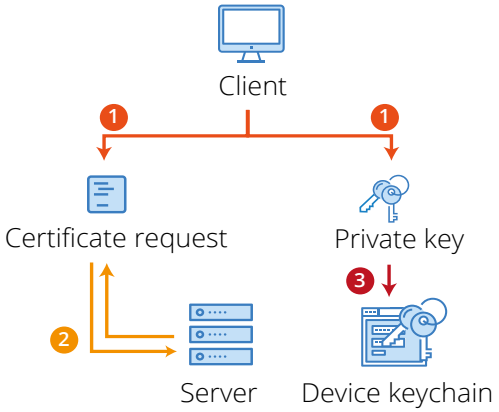
We’re investigating how a CSR approach here could help in such edge-cases at least to allow new share again. We do however encourage users to make sure to not lose access to all their devices as well as their recovery mnemonic at the same time.

Illustrations

How to create and sync identity

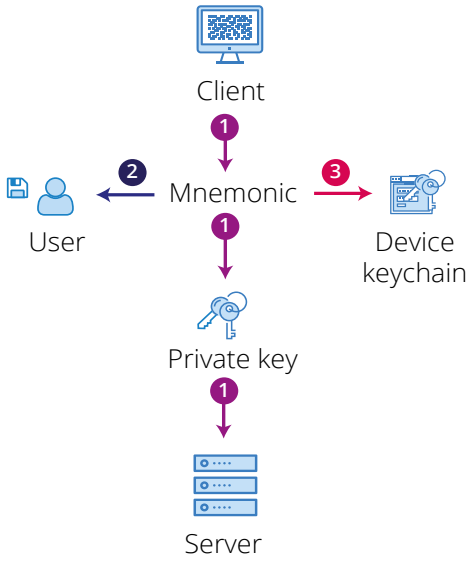
1

1. Client generates a new X.509 certificate request and private key.
2. Certificate gets signed by server.
3. Private key is stored in keychain of device.



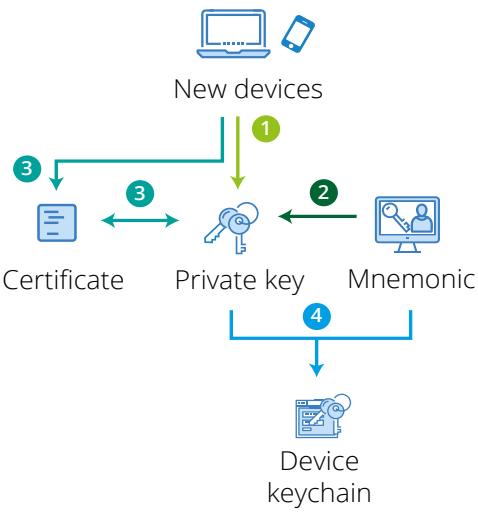
2

1. Client encrypts private key with 12 word mnemonic and uploads it to the server.
2. Mnemonic is displayed to users and they are asked to store it.
3. Mnemonic is stored in keychain of device.



3

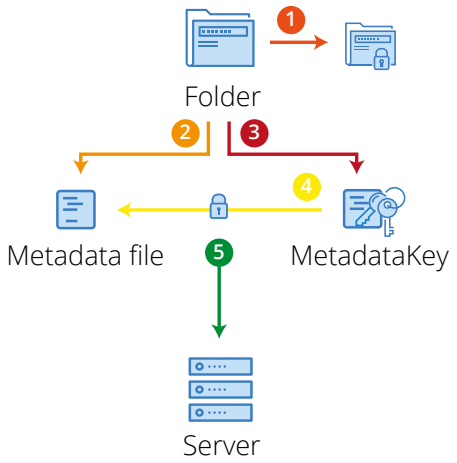
1. New devices download the encrypted private key.
2. Private key gets decrypted with the 12 word mnemonic from the user.
3. New devices check if private key belongs to certificate.
4. Private key and mnemonic are stored in the keychain of the device.



How to encrypt folders and add files

1

1. Create folder and mark to server as encrypted.
2. Generate metadata file.
3. Generate metadataKey, encrypted to all public keys that have access to the folder.
4. Use metadataKey to encrypt all values in metadata file.
5. Store encrypted metadata on server.

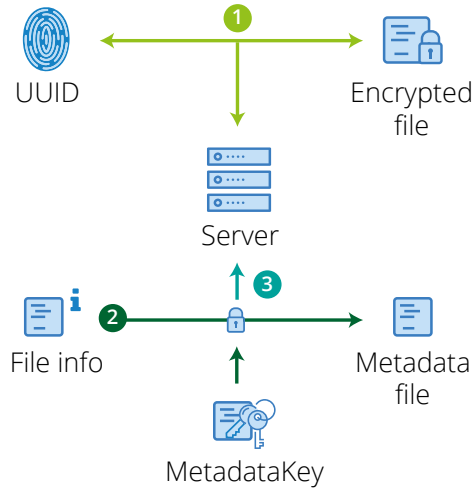


2

1. Generate new 128bit encryption key for file.
2. Encrypt it using AES/GCM/NoPadding.

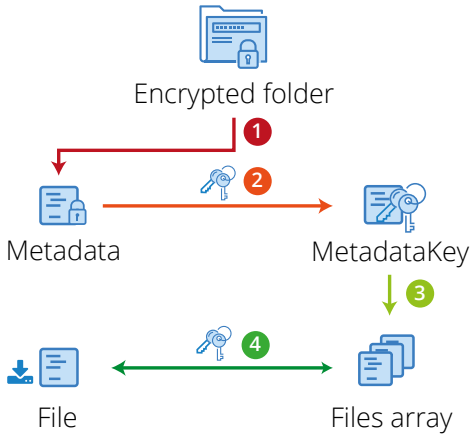
3

1. Generate random identifier (UUID) and upload encrypted file, using random identifier as file ID.
2. Add new file info to files array in metadata file, encrypted with metadataKey.
3. Update metadata on server.



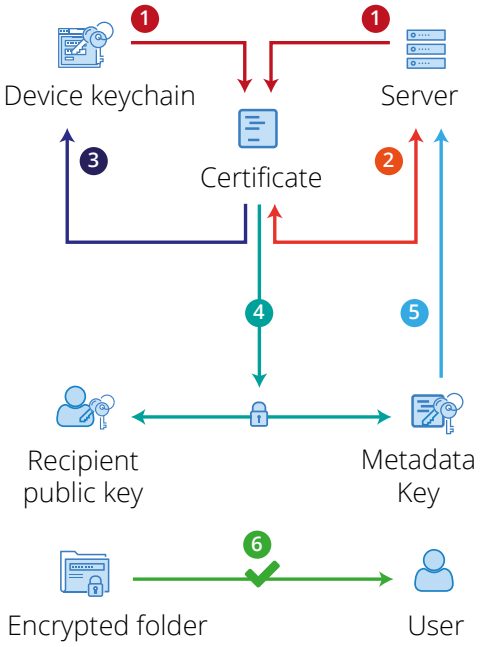
How to access encrypted files

1. Download metadata of encrypted folder.
2. Use private key to decrypt metadataKey.
3. Use metadataKey to decrypt to files array.
4. Download the files and decrypt them using 128bit AES/GCM/NoPadding using keys from the files array.



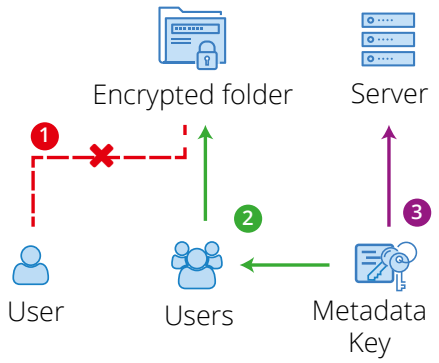
How to share encrypted folders

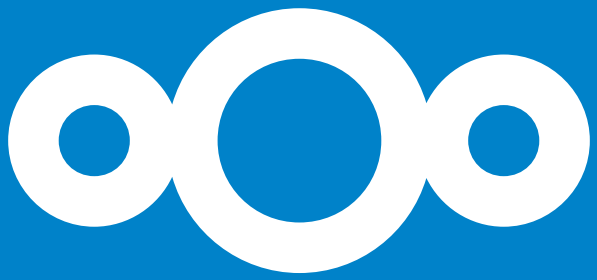
1. Check for certificate of specified user ID or download it from the server (Trust On First Use / TOFU).
2. Verify if the certificate is issued by the server.
3. Store user certificate locally.
4. Re-encrypt metadataKey to recipient public key.
5. Upload updated metadata to server.
6. Share folder with user through OCS sharing API.



How to remove users from encrypted folders

1. Unshare encrypted folder with user through OCS sharing API.
2. Generate new metadataKey and encrypt to everyone who now has access.
3. Upload metadata to server.





Nextcloud GmbH
Kronenstr. 22A
70173 Stuttgart
Germany

Email sales@nextcloud.com
Phone +49 711 896656-0
Fax +49 711 896656-10

nextcloud.com