STRATO

HiDrive

Client-side Encryption

Table of Contents

1 Concept

HiDrive customers shall be enabled to encrypt their data on the client-side and transfer only encrypted data. This protects their data from access by the storage provider. The encryption is done at file level and implemented by a cryptographic library. The master key and thus the control of the data remains with the client.

All formats, specifications of the algorithms and the type of key management should be publicly available. The source code for the encryption and transmission should be open, so it can be reviewed and compiled.

In the encrypted view, there is a key file for each file and folder. It contains the ciphered key for the respective resource. So the number of files is at least doubled compared to the un-encrypted view. These individual keys allow the targeted sharing of files and folders.

1.1 Hierarchical Keys

File- and Directory Keys are stored as separate files in the file system. File Keys are stored next to the respective files, Directory Keys are stored in the respective directory.

The result is a hierarchy. The master key (User Key) decrypts the highest Directory Key. A Directory Key decrypts the names and keys of all files and folders contained in a directory. A File Key decrypts the contents of a file. An example of such a hierarchy is in section 1.3. The following keys are used in the process.

File Key

A File Key contains a key to symmetrically encrypt and decrypt file contents. Here it is a 256 bit AES key. The File Key is itself encrypted with the Directory Key of the comprising directory or a Share Key.

Directory Key

In an encrypted directory the names of the directory entries are unrecognizable. The number of entries, their time stamps and sizes are visible.

A Directory Key contains keys to symmetrically encrypt and decrypt the names of included files and folders as well as file and directory keys. These are 256 bit AES keys. A Directory Key is itself encrypted with the Directory Key of the comprising folder or a Share Key. The Top Level Directory Key in the hierarchy is always asymmetrically encrypted with the User Key (the public key).

User Key

The User Key is an asymmetric key. Here it is an elliptic curve (EC) key pair for Curve25519. The private key decrypts the highest Directory Key. It enables the user to recursively read all lower directory keys and thus all file names and contents. It is generated with OpenSSL. This key remains with the user and can be stored in a password protected format.

If this master key is lost, all data is lost. It is recommended to save copies.

Share Key

A Share Key is an asymmetric key. Here it is an elliptic curve (EC) key pair for Curve25519. The private key decrypts the File- and Directory Keys that were created for a share. It is also generated with OpenSSL. For each share account there is a separate key. Only the creator and the receiver of the share know it.

1.2 Sharing

Each file has an associated File Key that is encrypted with the Directory Key. When a file is shared, the existing plain File Key is asymmetrically encrypted with the public key of the share receiver and also stored next to the file.

When a folder is shared, the existing plain Directory Key is asymmetrically encrypted with the public key of the share receiver and also stored within the folder.

If the receiver's public key is known, the shared resource key can be encrypted directly with the public key.

If the receiver's public key is unknown, a new Share Key can be created. It is the user's responsibility to transfer this new key securely and apart from the share link and password. The user can also choose to transfer the password protected key via HiDrive as explained in section 5.

1.3 Example Hierarchy

Example of a directory structure with the presented keys.

root	
🗀 public	
🗀 pics	
🗀 D_HDTLRN_lqToZ0mw8'8P9Hws#610.c	encrypted folder for multiple users
🗎 {B72C...DF62}.$ap5W.0.k	Top Level Directory Key (User Key encrypted)
🗎 {B72C...DF62}.Gc_0e.0.k	
🗎 {B72C...DF62}.F'hPF.0.k	
🗎 A15EPiaB%^o.0.d	encrypted file
🗎 A15EPiaB%^o.-.0.k	associated File Key (Directory Key encrypted)
🗀 users	
🗀 alice	
🗀 documents	
🗀 D_HDTLRN_@EWhXH}4Obe6d=Akl652.c	encrypted folder
🗎 {B72C...DF62}.$ap5W.0.k	Top Level Directory Key (User Key encrypted)
🗎 A%MO~KFL8A~.0.d	encrypted file
🗎 A%MO~KFL8A~.-.0.k	associated File Key (Directory Key encrypted)
🗎 A#9y]{.0.d	shared encrypted file
🗎 A#9y]{.-.0.k	
🗎 A#9y]{.sjxemdnH.0.k	File Key (Share Key encrypted)
🗀 A8 DeH^MZp)q8tDK.0.d	encrypted subfolder
🗎 B2Hl$...ewG}v.-.0.k	Directory Key (Directory Key encrypted)
🗎 AW(m(zPti]Q.0.d	
🗎 AW(m(zPti]Q.-.0.k	
🗀 A'4Sk0o``1l5EuXe.0.d	shared encrypted folder
🗎 BdjsQ...;w5g2.-.0.k	
🗎 CHEB3(Ep0L_1(#.0.k	Directory Key (Share Key encrypted)
🗎 AV%]}mp^Cl}.0.d	
🗎 AV%]}mp^Cl}.-.0.k	

2 Cryptographic Algorithms

All presented algorithms are implemented in the OpenSSL library (current version 1.0.2p). They are described and justified in this section. The selection of methods may be extended or changed in the future.

2.1 File Data

File data is symmetrically encrypted with

AES-256

in CTR-Mode

We encrypt file data in 4064 byte blocks. For each plain text block and its position we calculate a 32 byte message authentication code (MAC) that is prepended to each ciphered block. The first 16 bytes of the MAC are used as an initialization vector (IV) for the encryption of this block. Since the block counter is part of the authenticated message, identical plain text blocks receive different IVs and result in different cipher texts. The last block of a file is special and can be less than 4064 bytes in size.

This way the encryption and decryption of data blocks is deterministic and parallelizable. Determinism is required for HiDrive's synchronization algorithm to work correctly, i.e. two clients get identical results when encrypting the same file with the same key. Parallelization is essential for large files that are downloaded and uploaded in chunks.

Each file has its own randomly generated key and nonce. Copies of a file have different ciphers.

2.2 Key Data

Key data is symmetrically encrypted with

AES-256

in CTR-Mode

For the encryption of keys we use AES-256 in counter (CTR) mode.

Each key is randomly generated and encrypted with a random nonce.

2.3 File- and Directory Names

File and folder names are symmetrically encrypted with

Blowfish using a 256 bit key

in CBC-Mode

padded with zeros (similar to ISO/IEC 7816-4)

1st pass forward, 2nd pass backwards

For the encryption of file and folder names we use Blowfish with a 256 bit key in cipher-block chaining mode (CBC). It is applied forward on the original name and then backwards on the result with different initialization vectors (IVs). Afterwards the ciphered names are Base85-encoded (see section 3.1).

Stream cipher modes such as the cipher feedback mode (CFB) could not be used because there are names shorter than the block size. Those do not benefit from the cipher feedback and similar names lead to similar ciphertexts. We chose cipher block chaining mode – CBC.

There are restrictions on the overall path length on some systems. Long individual path components limit the nesting depth of directories.

To prevent waste while padding file names we opted for a 64 bit block cipher. Since this is the encryption of directory entries, data will not amount to the sizes necessary for two blocks to become identical by chance (birthday paradox). Candidates were Blowfish, CAST, 3DES, IDEA, and RC5. Blowfish is free, available in OpenSSL and offers key sizes of up to 448 bits but is also the oldest of the algorithms. It is widely used and the best public cryptoanalyses only apply to reduced numbers of rounds. CAST5 is far less known. IDEA and RC5 are not recommended because of better known cryptoanalyses. 3DES is still recommended as a legacy cipher but offers smaller key sizes.

Each directory has a randomly generated key and IV. They allow the names in a directory to be listed efficiently. No two directory entries have the exact same name. By processing the names forward and backwards small differences propagate and produce drastically different results.

2.4 Top Level Directory Key, shared File- and DirectoryKeys

The Top Level Directory Key as well as shared File- and Directory Keys are asymmetrically encrypted with an Elliptic Curve Integrated Encryption Scheme (ECIES) based on Curve25519.

RSA was previously considered for the asymmetric encryption, but the involved keys were too large for the import to mobile devices via QR-Codes. Elliptic curve cryptography allows for smaller keys while offering the same level of security. The next section describes the ECIES.

Asymmetric Encryption and Decryption

For the encryption using elliptic curves the following choices were made:

Key agreement algorithm (KA):	Elliptic Curve Diffie-Hellman (ECDH)
Key derivation function (KDF):	SHA-512
Encryption (enc) / Decryption (dec):	AES-256 in CBC-Mode with CMS padding (RFC 5652) and random IV
Message authentication (MAC):	HMAC with SHA-256

The user's key pair is (v, V) with $V = v * G$ (G is the generator of the group). Encryption transforms a message m to a cryptogram (U, t, c) with the user's public key V. It is implemented in these steps:

1. generate an ephemeral key pair $U = u * G$
2. $KA(u, V) \longrightarrow$ shared secret
3. KDF(shared secret) $\longrightarrow k_{mac}, k_{enc}$ (i.e. two keys)
4. $enc(m, k_{enc}) \longrightarrow c$ (cipher text)
5. $MAC(c, k_{mac}) \longrightarrow t$ (a tag)
6. Return (U, t, c)

Decryption reverses the process. It transforms a cryptogram *(U, t, c)* to the plain message *m* with the user's private key *v*. It is implemented in these steps:

1. Read the cryptogram *(U, t, c)*
2. *KA*(v, *U*) ⟶ shared secret
3. *KDF*(shared secret) ⟶ k_{mac}, k_{enc}
4. *MAC*(*c*, k_{mac}) ⟶ *t'* (fail if $t' \neq t$)
5. *dec*(*c*, k_{enc}) ⟶ *m* (plain text)
6. Return *m*

The library offers methods for the import and export of private and public keys.

2.5 Password Protection of User Key and Share Keys

The User Key pair as well as Share Key pairs should be protected with a password. They are symmetrically encrypted with

AES-128

in CTR-Mode

with IV and key from Scrypt(N=16384, r=8, p=1) over the password.

The length and complexity of the password is not restricted.

Section 5 describes a method for storing password protected Share Keys in HiDrive.

3 Formats

3.1 File- and Directory Names

Base85 Encoding

Encrypted file names can contain control characters. The file names should still be representable on all platforms. The result of the encryption is therefore Base85-encoded. The alphabet contains the ASCII characters 0x20 to 0x7e with the exception of reserved characters in Windows **" * / : < > ? \ |** and the baseline dot (**.**) which is used as a delimiter here.

Four bytes can be represented by five characters ($2^{32} \approx 4.3$ million $< 85^5 \approx 4.4$ million). The encoding has a small space advantage over Base64.

Encrypted file names

The name of a file or folder is made up of a single character indicating the type of encoding, and the encoded name itself. Here the first character is '**A**' for the following method:

Blowfish in cipher-block chaining mode (CBC) is used for the encryption of names. It is applied forward and backwards. The result is Base85-encoded as described above. If the original name is longer than 164 characters, then the encoded name is longer than 205 characters. In this case the encoded name is cut of and the remainder is written into a separate overflow file. Its name is composed of the cut-off name followed by a dot (not part of the alphabet) and the suffix '**n**'. The file name itself is ends with '**.0.d**'.

The name of a key file is composed of the unrecognizable object name, a dot, a Key ID, '**.0.**', and the suffix '**k**'. The Key ID specifies which key was used to encrypt the key. It is '**-**' for the Directory Key of the enclosing folder. It is the Base85-encoded account ID for a User Key (max 40 characters) or the Base85-encoded Sharelink ID for a Share Key.

The component '**.0**' is reserved for future extensions. The key name is therefore a maximum of 251 characters long.

Component	A	<crypt. Base85-encoded Name>	.	<Key ID>	.0.k	total
Length	1	205	1	40	4	251

Example: File name with 207 characters

```
4f50 5553 202d 2057 6972 6b75 6e67 7376 6f6c 6c65  OPUS - Wirkungsvolle
7320 4368 616e 6765 204d 616e 6167 656d 656e 7420  s Change Management
696e 2041 6268 c3a4 6e67 6967 6b65 6974 2076 6f6e  in Abh..ngigkeit von
2073 6974 7561 7469 7665 6e20 416e 666f 7264 6572   situativen Anforder
756e 6765 6e20 206f 7267 616e 6973 6174 696f 6e61  ungen  organisationa
6c65 2056 6572 c3a4 6e64 6572 756e 6773 7072 6f7a  le Ver..nderungsproz
6573 7365 2069 6d20 5370 616e 6e75 6e67 7366 656c  esse im Spannungsfel
6420 766f 6e20 6265 7472 6965 626c 6963 6865 6e20  d von betrieblichen
566f 7261 7573 7365 747a 756e 6765 6e20 756e 6420  Voraussetzungen und
556d 7765 6c74 616e 666f 7264 6572 756e 6765 6e2e  Umweltanforderungen.
6874 6d2e 677a 65                                  htm.gze
```

After the forward and backward encryption with a Directory Key

```
eacd 3370 76be 97fa 3d0f f981 1f3b 5d67 609a b181  ..3pv...=....;]g`...
93ff 0041 1dfe 791a 4dfb 1279 af78 2440 87f3 894d  ...A..y.M..y.x$@...M
435d 0d13 c626 5eca 3c71 9479 427b 6ef9 480f 11e9  C]...&^.<q.yB{n.H...
73c1 f2e7 2f10 04ae b359 d388 e09c 1238 a81d 23f6  s.../....Y.....8..#.
9108 ac84 9fab edeb ab1c 61af 88c5 51f9 a8b3 51cd  ..........a...Q...Q.
53f6 ddf2 74a4 6ac6 91f5 34d9 564e eab0 056a 4603  S...t.j...4.VN...jF.
4cf2 e970 711e 3246 3df3 5366 132e 954f d14e b362  L..pq.2F=.Sf...O.N.b
dde0 b9a1 941d 7562 4238 bb29 69e2 fb5a 9fdf 4c4f  ......ubB8.)i..Z..LO
1cf1 880e e286 5fd5 b380 b583 e724 fea5 1ee8 9a35  ......_......$.....5
47c1 30b7 f682 d85c ea7f df87 8d5e e383 4590 9939  G.0....\.....^..E..9
4a18 e32c 9ca4 52                                  J..,..R
```

The Base85-encoding of this has 260 characters.

```
=diB7cD{STJrDVTA3I$VV4AUklm7rg9sYS5P5TmguXrRthx3U{LtPCM!wK3MJaLqHLVIrc
NdmR{bHVcGFAxN+vsu$K,G7aTs2wBrkqE4WpR4Wbs~lmkh`aL'sIyVcQ}(5Rbfjv;k[Ym#
R +_#1 ^hEO Db'aUL=8J[Zp#6E2lc&N42s+Qc,QlpS@qLO8oAY2y1[pWjSR9Px)^,'Y)-
vw(dO;Oq569^X4iN5L@)^JY_4=VMydjb7t}MUa]}N(LoToTO3
```

The truncated names (to 205 characters) are:

File name:	`A=diB7c...pWjSR9Px)^.0.d`
Name Key File (Directory):	`A=diB7c...pWjSR9Px)^.-.0.k`
Name Key File (User):	`A=diB7c...pWjSR9Px)^.HDqaWHZ&OeE,2GP00n.0.k`
Name Overflow File:	`A=diB7c...pWjSR9Px)^.n`

The content of the overflow file is:

```
,'Y)-vw(dO;Oq569^X4iN5L@)^JY_4=VMydjb7t}MUa]}N(LoToTO3
```

Directory Key Name

The name of a Top Level Directory Key is the designated prefix `{B72CB30E-D9C5-4475-8BD2-664E03A5DF62}` and the extension `.<KeyID>.0.k`. KeyID again specifies the key that was used to encrypt the Directory Key. It is either a Base85-encoded account ID or '**-**' for the Directory Key of the parent folder.

Normal Directory Keys start with the prefix '**B**' followed by a Base85-encoded SHA-224 hash over the unencrypted name and the parent key. This has the advantage that during a rename or a move the new and the old key may lie side by side. In case of failure the old key will continue to work.

Directory Keys of shared folders start with the prefix '**C**' followed by the Base85-encoded share id and '**.0.k**'.

Top Level Directory Name

The name of a top level directory starts with '**D_HDTLRN_**' followed by 20 random characters from the Base85 alphabet and the suffix '**.c**'. The plain name is contained in the key file and should be visible in the unencrypted view. To prevent key exchange attacks the key file is cryptographically bound to this random name.

3.2 File- and Directory Keys

The file format of a File- or Directory Key is as follows:

Field	Hex Value	Size (bit)
Magic	48 44 4B 46	32
Version	07 00	16
Type	00	8
Algorithm	varies	8
Reserved	00 00 00 00	32
Nonce_key	varies	128
Nonce_content	varies	128
IV_name_forward	varies	64
IV_name_backward	varies	64
Cipher size (in bytes)	varies	16
Cipher <Keys + Hashes>		varies

All numbers are stored in little-endian format. A key file is 190 to 559 bytes in size. The file signature (magic) is "HDKF" and the version is 7. Type is reserved and zero.

Algorithm describes what method was used to encode the key data.

0 AES-256 in CTR Mode (key is encoded with a Directory Key)

1 *obsolete:* RSA with 4096 bit key

2 ECC with Curve25519 (key is encoded with a User or a Share Key)

The following 4 bytes are zero and reserved for future use.

Nonce_key is the nonce for the symmetric encryption of this key. Nonce_content is the nonce for file data encryption. IV_name_forward and IV_name_backward are the initialization vectors for name encryption.

The next 2 bytes (Cipher size) indicate the size of the cipher text in bytes. For AES that are 128 bytes, for elliptic curve cryptography (ECC) at least 257 bytes.

Cipher is the cryptogram of

- AES-256 key for data and keys,
- AES-256 key for message authentication codes,
- AES-256 key for file and directory names,
- the first 128 bits of a SHA-256 hash over both keys and the file or folder name,
- the first 128 bits of a SHA-256 hash over both keys only, and
- optionally the plain name of a top level directory.

Keys and hashes are encrypted either with the Directory-, User- or Share Key. The hash enables us to quickly detect whether the key belongs to this item and whether it was correctly decrypted or not.

3.3 User Key and Share Keys

A User Key is a file the user keeps to himself. Share Keys remain with the share user. They are elliptic curve (EC) private keys. The file format is as follows:

Field	Hex Value	Size (bit)
Version	2	4
Type	0	4
Cipher <Key + Hash>	varies	320
Salt	Varies	64

Version is 2. Type 0 denotes a private key. Cipher is the AES_128_CTR cryptogram over the plain Curve25519 private key (a 32 byte number) and the first 64 bit of its SHA-256 hash. IV and Key for the encryption are derived from the password using Scrypt. If no password was provided, Cipher is the unencrypted private key and its hash.

The library can export public keys. These are always unencrypted. The file format is:

Field	Hex Value	Size (bit)
Version	2	4
Type	1	4
Point Representation	04	8
P_x	varies	256
P_y	varies	256

Version is 2. Type 1 denotes a public key. Point Representation is 0x04 for the uncompressed representation of an elliptic curve point according to X9.62. P_x and P_y are the coordinates of the public key (a point on the curve).

4 File System Operations

This section outlines the necessary steps for specific operations

4.1 Files

Creating a File

File Data ← File Key ← Directory Key

When creating a file in an encrypted folder, its name is encrypted with the Directory Key, Base85-encoded and truncated if necessary. This name is used for the file itself, its key file and its overflow file. An AES-256 key is generated and the file content is encrypted with it. The key is then encrypted with the Directory Key and saved as a File Key.

Reading a File

User Key → Directory Key → File Key → File Data

A normal user starts a session with the User Key. When changing to a folder, the prefix '**D_HDTLRN_**' in the directory name indicates that the encrypted hierarchy starts here.

The Top Level Directory Key is loaded and decrypted with the private User Key. Then the names and keys of the directory entries can be decrypted. The File Key of the desired file is loaded and decrypted with the Directory Key. The contents of the file can now be loaded and deciphered in whole or in individual blocks.

Reading from arbitrary offsets is possible but it is possible that data from blocks surrounding the start and the end of the range is required. The library provides functions for calculating the offsets and sizes of those blocks.

4.2 Directories

Listing a Directory

User Key → Directory Key → … Directory Key → Names

Calculate the encrypted path by encrypting one component at a time with the respective Directory Key. List the directory. Check each entry matching '***.d**' if its length requires an overflow file and read that. Decrypt each entry matching '***.d**' with its potential overflow file using the Directory Key.

Moving a Directory

Encrypt the new path first and calculate the name of the target Directory Key. Re-encrypt it with the target parent Directory Key and put this new key with its new name in the old source Directory. Now perform the move. Delete the old Directory Key from the target.

Remove the potential overflow file on the source side and create the overflow file on the target side if necessary.

5 Sharing

For sharing a resource we create a new key pair (Share Key). The user can rely on the HiDrive API to store the password protected Share Key.

The key (File- or Directory Key) of the shared resource is decrypted, re-encrypted with the public Share Key and stored together with the resource. For example a file will then have an additional File Key with the suffix `.HdqaWHZ&OeE,2GP00n.0.k`. A folder will contain an additional Directory Key with the name `CVKrejW,rl&Hgq%.0.k`.

The receiver of the share will need the share link and the password that must be transmitted securely. The share can then be accessed using the client software.

5.1 Share Key in HiDrive

Sharing a Resource

User Key → Directory Key → Directory Key ← public Share Key

First the client software generates a salt value and the user is prompted for a password. From the salt and the password we derive a share access key (SAK) and a cipher key (CK) using Scrypt.

Then it generates the Share Key and encrypts it with the CK. The result, the protected share key, is passed to the API call POST /share along with the salt and the SAK.

The Directory Key to the shared file is encrypted with the public Share Key and additionally stored in the folder.

Accessing a shared Resource

private Share Key → Directory Key → Directory Data

The receiver of a share will get a link and password. With the API call GET /share/info he also gets the salt value and can now derive SAK and CK.

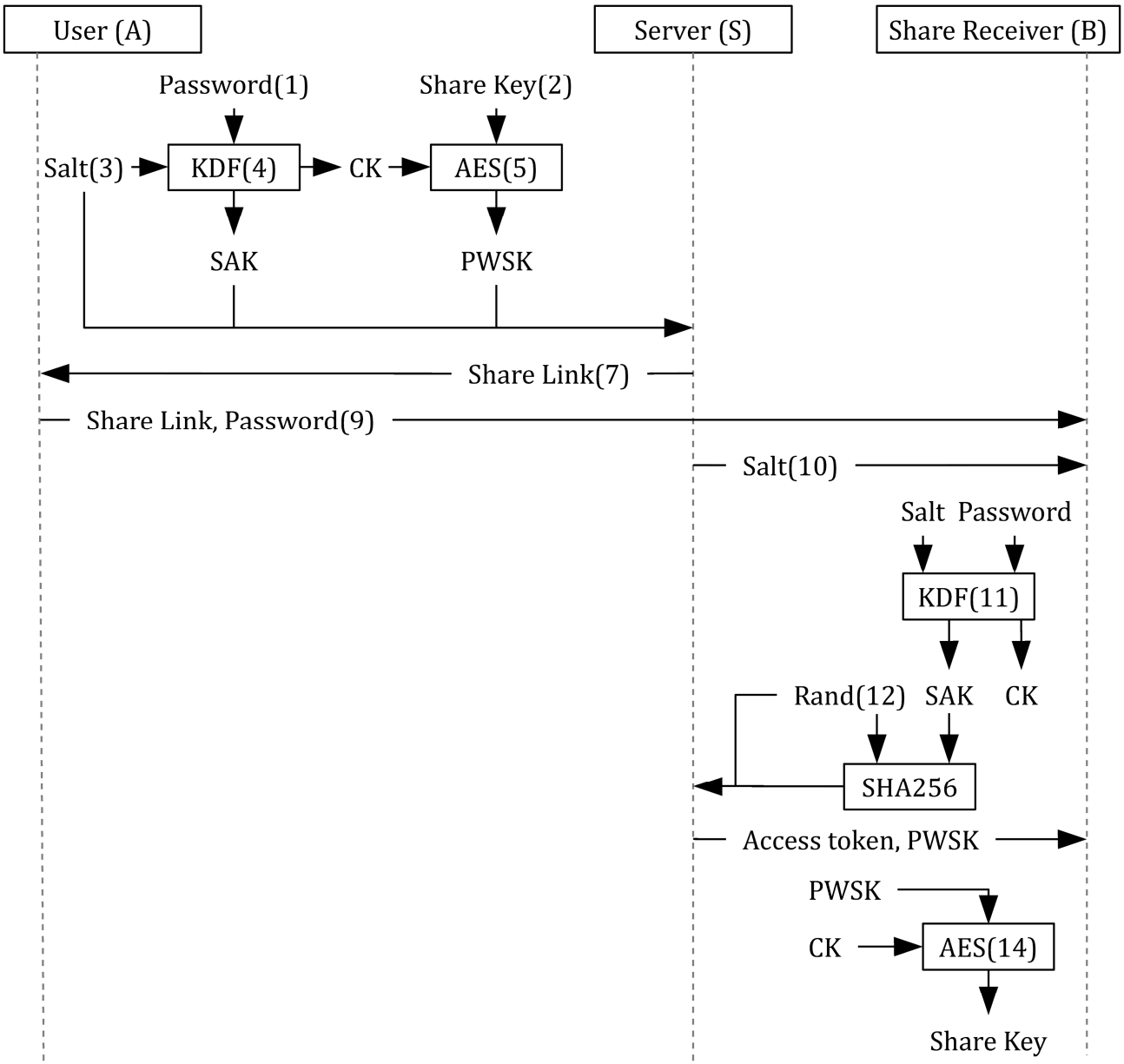
The receiver authenticates to the API as follows: generate a random value and concatenate it with the SHA256 hash over the random value and SAK. The result is sent to the API via POST share/token.

The server responds with the protected share key, that the receiver can now decipher using CK. The share ID leads to the name of the Directory Key. The private Share Key decrypts the Directory Key and the contents of the folder can be read.

Detailed process of sharing an encrypted folder

There is the user creating the share with client software (A), the server (S) and the receiver with client software (B). This is about the case that the password protected private Share Key is stored on the server.

1. The user is prompted for a password PW.
2. A generates the asymmetric Share Key SK.
3. A generates a random value Salt.
4. A calculates kdf(PW, Salt) and derives two keys:
    1. cipher key CK to encrypt the Share Key and
    2. share access key SAK to authenticate the receiver against HiDrive.
5. A encrypts SK with CK. The result is the password protected share key PWSK.
6. A sends the request for share creation to the server. That is POST /share with the parameters Salt, SAK, and PWSK.
7. S stores these values and answers with a link to the share containing a share ID.
8. A re-encrypts the Directory Key with the public SK, calculates the new name and stores it within the directory.
9. A sends the link and the password PW to B, preferably via separate channels.
10. B calls GET /share/info and receives the Salt-value from the server.
11. B calculates kdf(PW, Salt) and derives the same CK and SAK
12. B generates a random value Rand, calculates SHA256(Rand, SAK) and sends both to the server (Rand, SHA256(Rand, SAK)). This construct prevents the server from sending an arbitrary salt, for which it could – with the help of a precalculated rainbow table – derive the password.
13. S performs the same calculation with Rand. If it yields th same result, S grants B access to the share and sends the PWSK.
14. B decrypts PWSK with CK. The result is the Share Key (pair).
15. B determines the name of the re-encrypted Directory Key from the link ID, loads and decrypts it with the private Share Key and is now able to recursively read the directory.



5.2 Expiration of Shares

The expiration of shares is not cryptographically enforced.

Shared resources keep their keys, even if the share has expired. The share receiver can no longer authenticate. But if he gains access to the encrypted data by other means (intrusion at the storage provider) he can continue to read the data once shared. This could be avoided by re-encrypting all of the shared resources.