DocuTrack

DocuTrack is a dapp for sharing and managing documents. You can upload documents and authorize people to access them with a few mouse clicks. Access to shared documents can expire or be explicitly revoked. In addition, you can ask other people to upload documents for you by simply sharing a link (no login required). Documents are transmitted and stored in encrypted form. The dapp can be used with any standard web browser, no plugins or extensions are needed and no passwords need to be remembered.

Such a dapp can only be realized on the Internet Computer (IC). It is the only blockchain network that can serve web content directly. Furthermore, its programming model enables such complex applications with privacy-preserving identity management fully on-chain. Last but not least, the IC provides low latency, efficiency and affordable storage facilities.

The documents are encrypted at all times, so no one–including IC node providers– but designated users can decrypt them.
With DocuTrack a user Alice who created an account with the dapp (using Internet Identity) can ask Bob to upload documents for her without having to create an account himself. This feature makes it very easy and secure for service providers (e.g., a client advisor or wealth manager) to request documents of any type (e.g., house deeds, passport pictures or log files) from clients or other third parties. Other document sharing apps require users to exchange public keys or other cryptographic material with which people typically struggle a lot.

Table of content

How is DocuTrack implemented and how does the encryption/sharing work?

The basic functionality of DocuTrack consists of two main components.

The frontend contains the functionality to

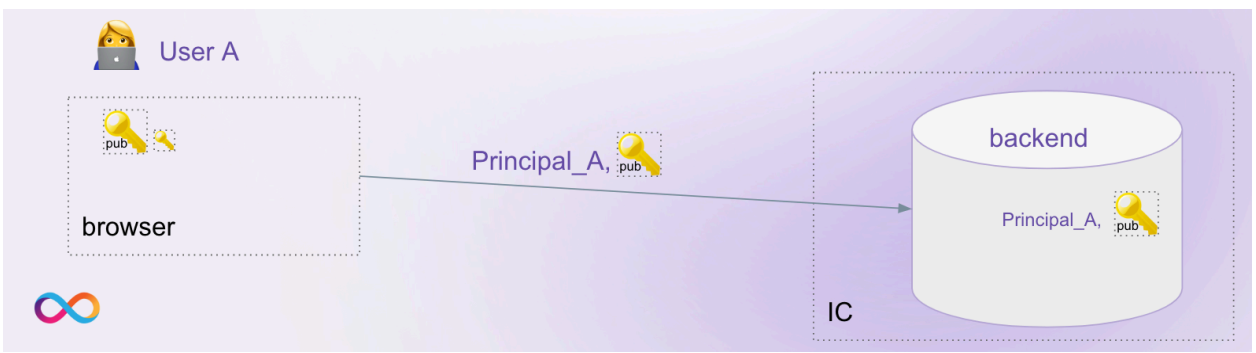
- Serve the user interface to the browser
- Create user and document keys and perform encryption

The backend contains the functionality to manage

- Registered users and their public keys
- Encrypted documents
- Document decryption keys encrypted for all authorized users for a document
- Uploading of new documents

User registration

Users are linked to their Internet Identity in the frontend after login, the corresponding principal will be used for calling backend API queries and updates.
After registration a public / private key pair is created for the user.
The public key is stored in the backend for each registered user, together with their name.

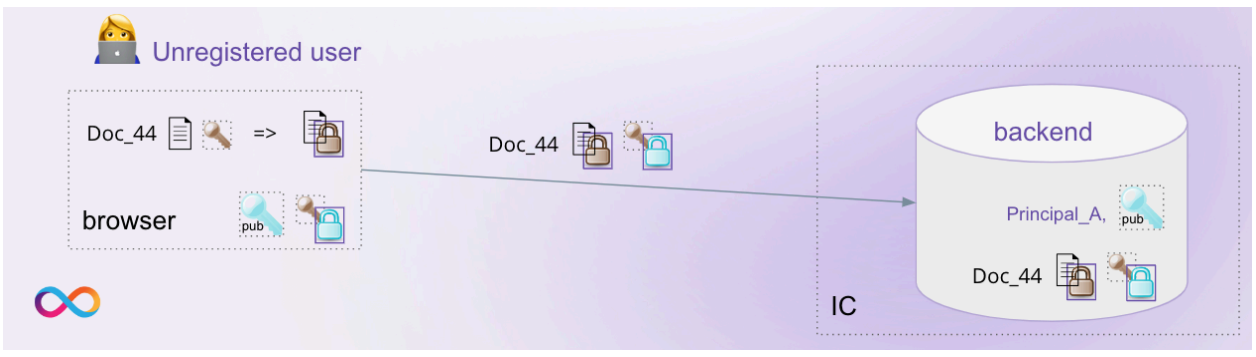


Document Upload

When a document is to be uploaded, a symmetric secret key for the document is created in the (potentially unregistered) user's frontend. In the example above, this user is called Bob. The document is encrypted with this secret key.

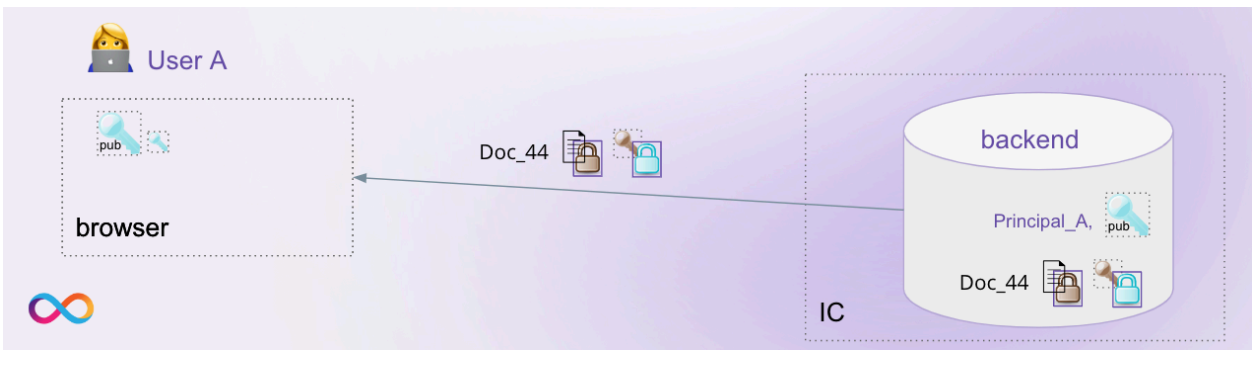
As a next step, the frontend obtains the public key of the registered user who created the request link, Alice, from the backend. The document's secret key is encrypted for Alice.

Both the encrypted document and the encrypted secret key are sent to the backend where they are stored.



Document Access

The Frontend of registered user Alice can get the encrypted doc together with the encrypted key from the backend canister. Using the private key, Alice's frontend can decrypt the document and process it.

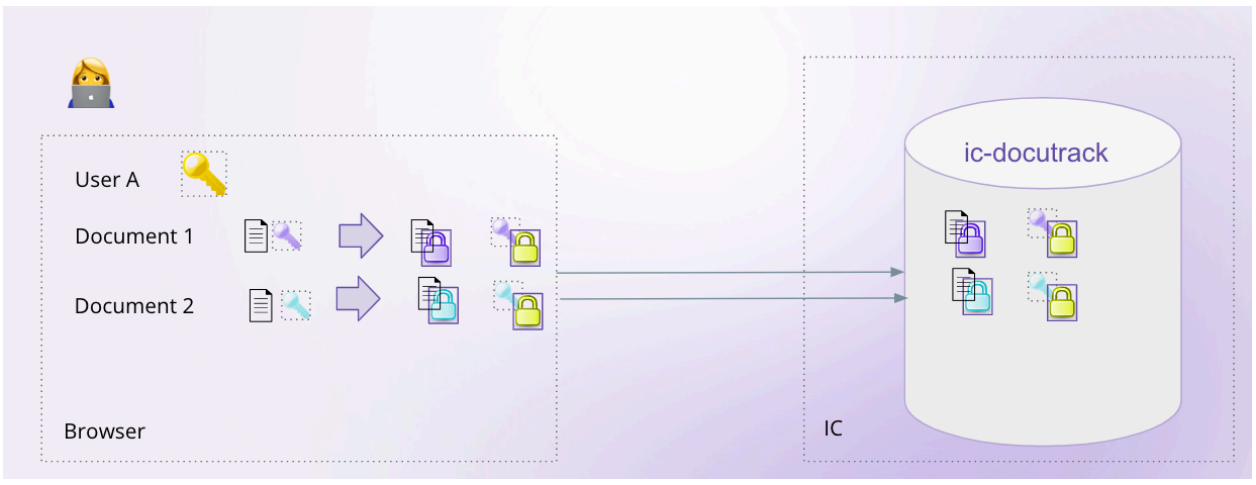


Cryptography

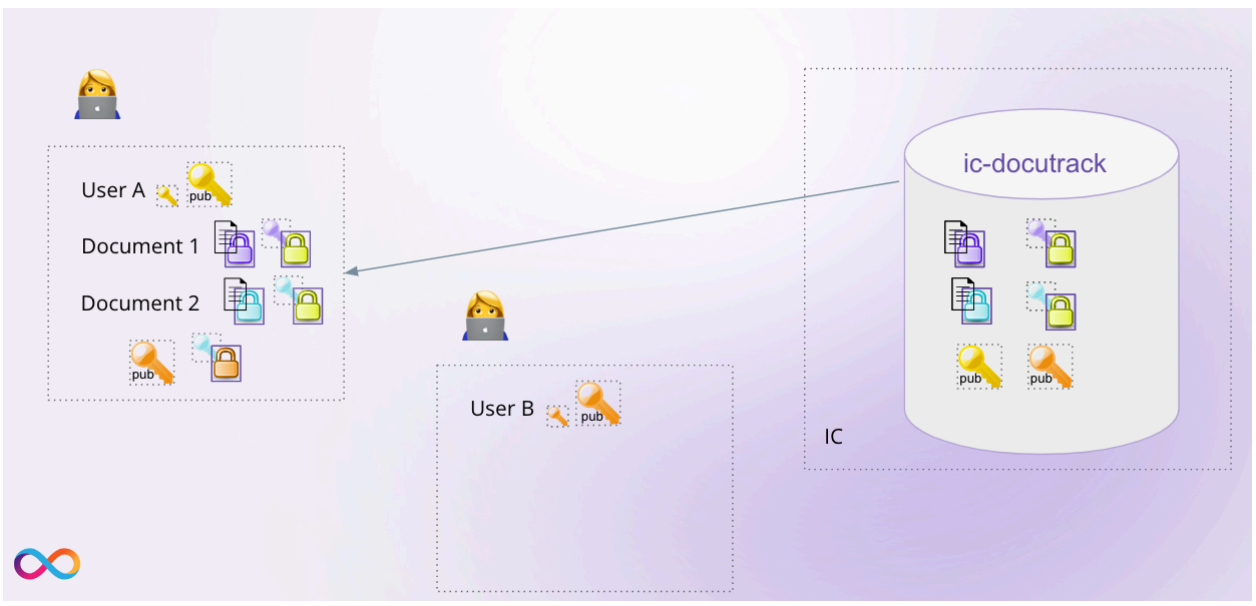
The creation of keys, encryption of documents and their decryption keys is done entirely on the client side. The dapp uses different kinds of keys (link to source code):

- Documents
  Symmetric AES-GCM secret key: used to encrypt and decrypt a document. This secret key is stored encrypted for one or several users in the backend.
- Users (II principals)
  RSA-OAEP public key: used to encrypt the symmetric AES secret key of a document the user should have access to. The public key for each registered user is stored in the backend.
  RSA-OAEP private key: used to decrypt the symmetric AES secret key of a document stored in the backend. Once the frontend decrypts the secret key, it can use this key for decrypting the corresponding document stored in the backend. The private key never leaves the client-side and is not stored anywhere else.

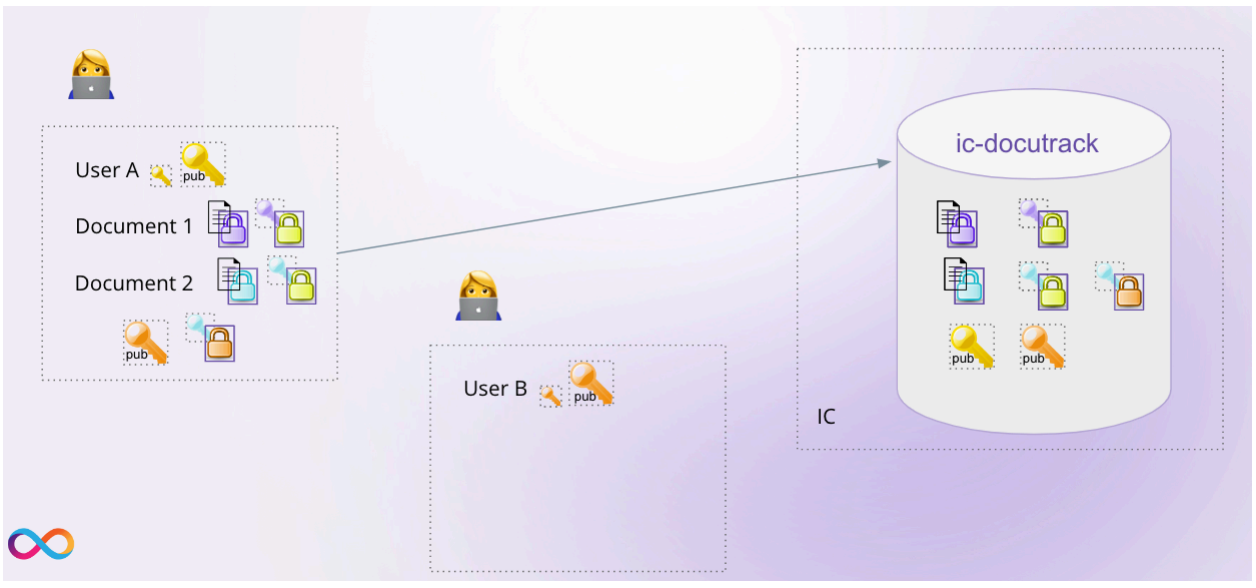
For each user and for each document separate keys are being used.



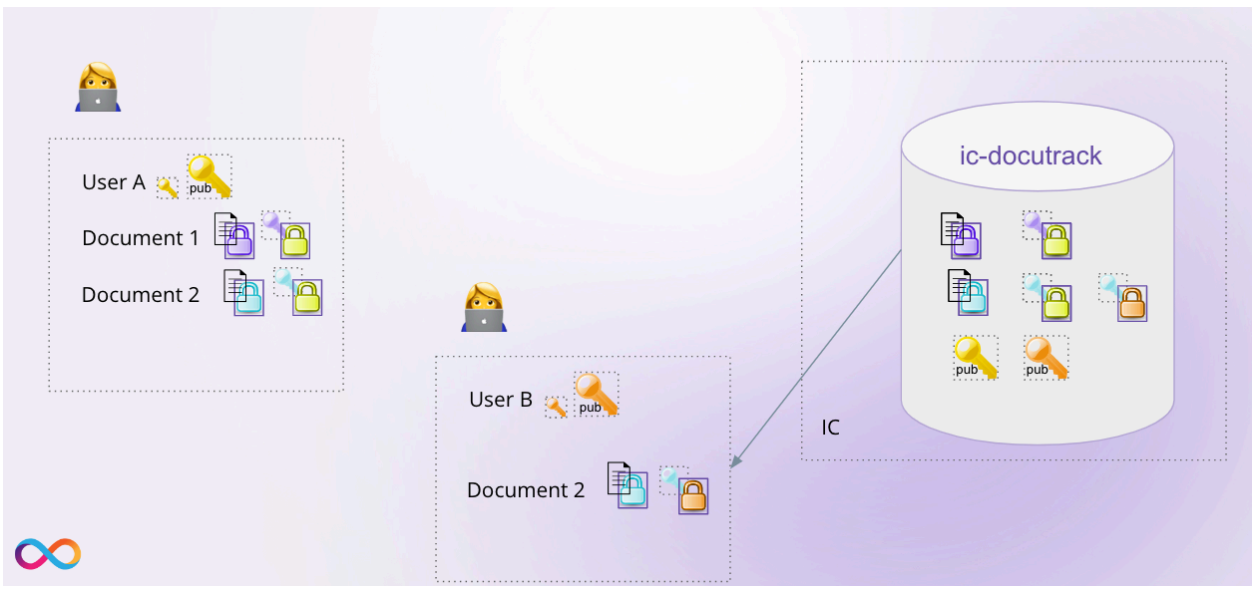
In order for Alice to share Document 2 with Bob (who needs to have an II to support this functionality), the secret key of Document 2 needs to be encrypted for Bob's public key.



This key is then added to the backend canister



Bob's frontend can now download the document together with the document key encrypted for Bob and decrypt it locally.



Considerations for Productization

Key Management

This dapp demonstrates the potential of sharing encrypted files on the IC. Do not use this code as is for sensitive data as there are the following issues that should be addressed before the dapp can be considered production-ready:

- Users may lose their notes if they accidentally clean the browser data (localStorage)
- The frontend re-uses the generated public- and private-key pair for every identity in the same browser. In a better implementation, this key pair should be unique per principal and not managed by the browser at all.
- Lack of key update: Given that the key used to encrypted the files is never refreshed, the privacy of the data is no longer guaranteed if an attacker learns this key (for instance, by corrupting the local storage of one of the users).
- The same user cannot access the docs in another browser because the decryption key will not be available there.
- Productization: The best solution for the first two bullet points is to apply threshold key derivation to ensure in a clean and robust way that the same key pair can be extracted for each principal, regardless of the machine and browser used to access the dapp. Until this feature is available, key management could be implemented with WebAuthn extensions, see e.g., link1, link2, link3. However, these approaches are probably rather brittle, due to lacking widespread support in browsers and HW. For the last point, key revocation and/or key rotation should be used.

Auditability

- Note that as developer of a dapp, you can make the source code publicly readable so everyone can see how keys are being destroyed after access has been revoked. Anyone can then check if the canisters really run the version that belongs to the shared source code.
- The current implementation of the backend canister does not store the user actions (requests, sharing, opening, downloading,...) in a way that can be verified independently.
- Productization: Before extending the current implementation, one would have to define what are the required trust assumptions and potentially apply an approach that maintains a tamperproof and externally auditable blockchain in a canister, see e.g., link4.

Data Management Aspects

- The poc supports files of a size of up to 100MB each and a total of about 400 GB of data.
  (cost for 1GB data on the IC is about 5$ per year)
- Productization: For more total space, a mechanism to deploy an additional canister if the current one is full and to access the right canister when retrieving the data must be implemented

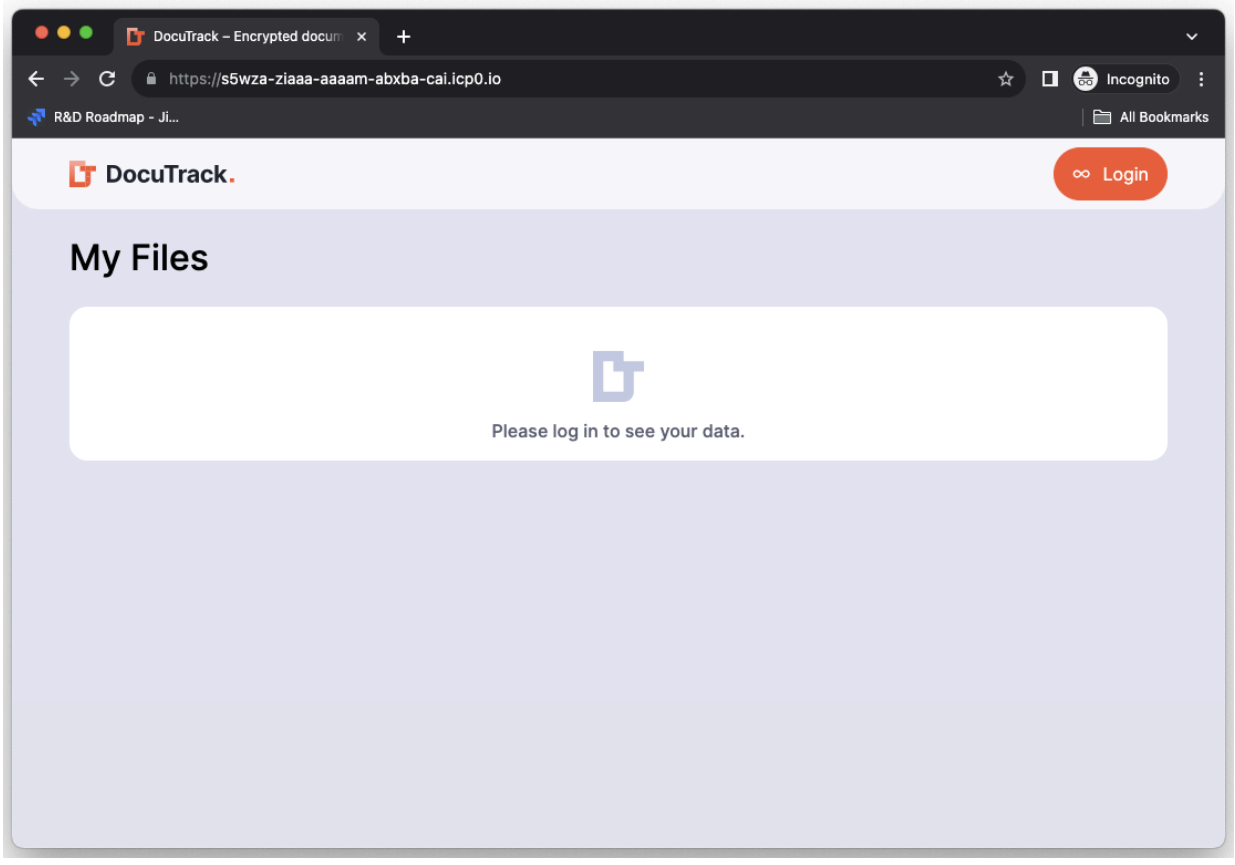
Ideas for product extensions/usability improvements

- Extend preview to include videos and PDFs could be shown in iframe
- “Select a file” could also act as a draggable target, same for the upload field itself (e.g., by making it a label?)
- Fix label for file upload, it says “fileNfile-selectorame” but the id of the field is: “file-selector”
- List files and request on the same start screen, segment the list into two parts
- Add an optional description field for the file you are requesting (instead of name only)
- Use AI to name the file you are uploading (recognize what is on the picture or document)
- Avoid overlays/popups (considered to be UX antipattern)
- Instead of sharing with a user, create a one-time link to send (add decryption key for logged in users opening it)
- If requests and files would be on the same overview, navigation would not be needed, instead just a nice call to action to create a request and upload a file (in place, not in a popup)
- When default email is set to browser, e.g., with gmail, the behavior when clicking on “send email” in file requests is unexpected and clunky
- Replace list of all users in sharing with more advanced search functionality (currently all users are shown + autocomplete)

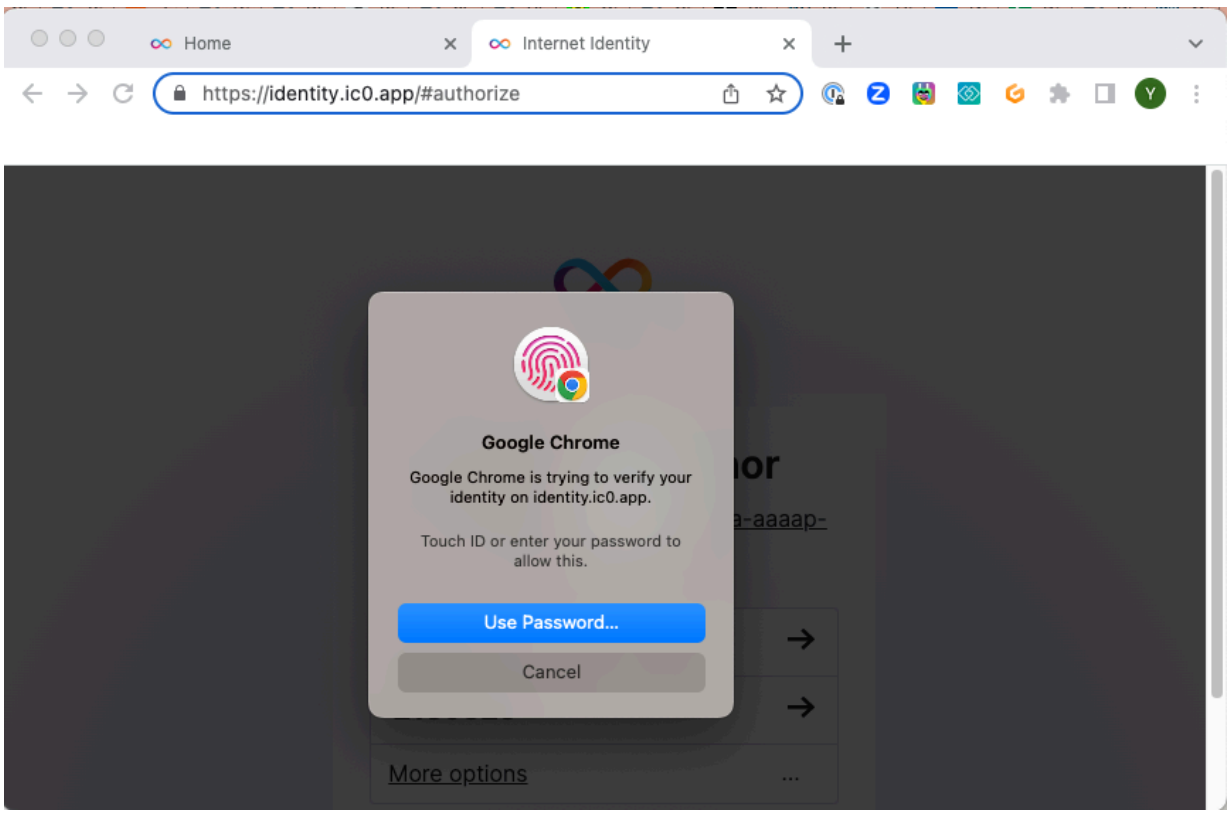
How can I use it?

You can try out DocuTrack on https://pxg7f-3yaaa-aaaap-qa6eq-cai.icp0.io

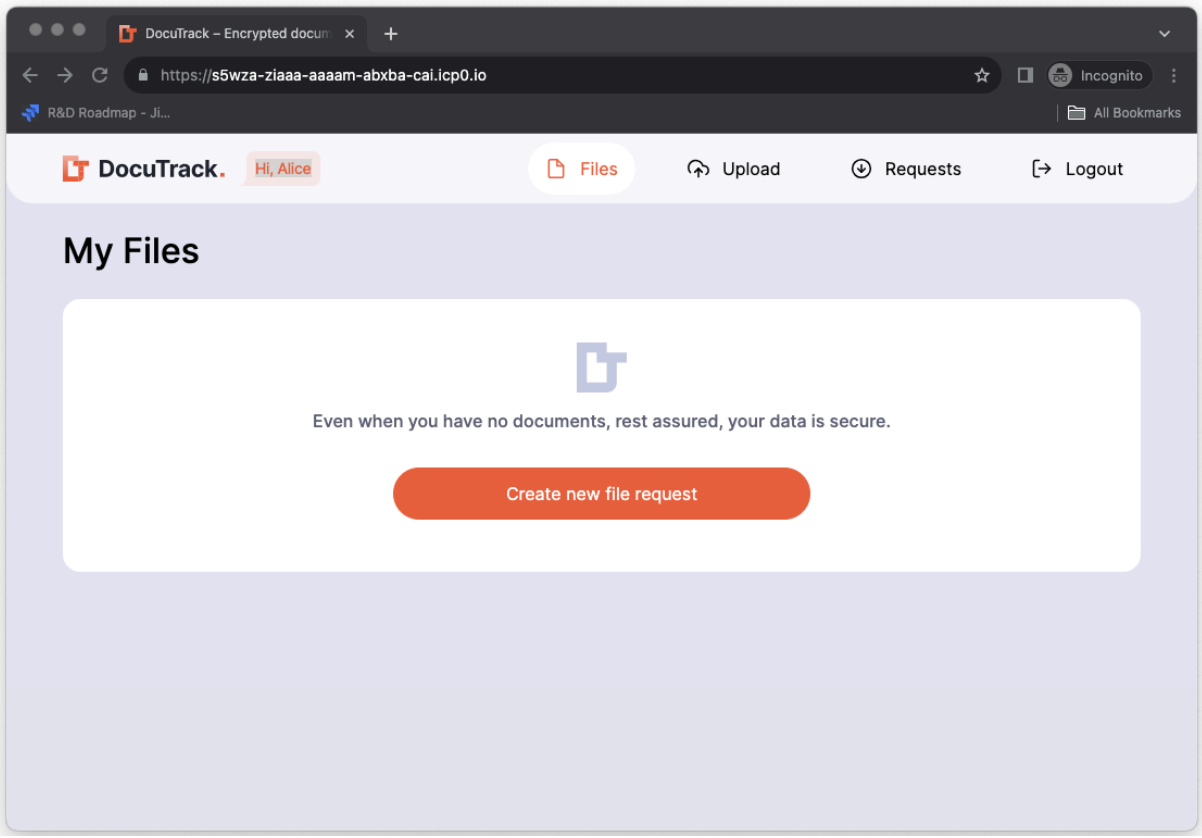
We'll walk through the example of Alice requesting a passport picture from Bob and then sharing it with Hans Peter.



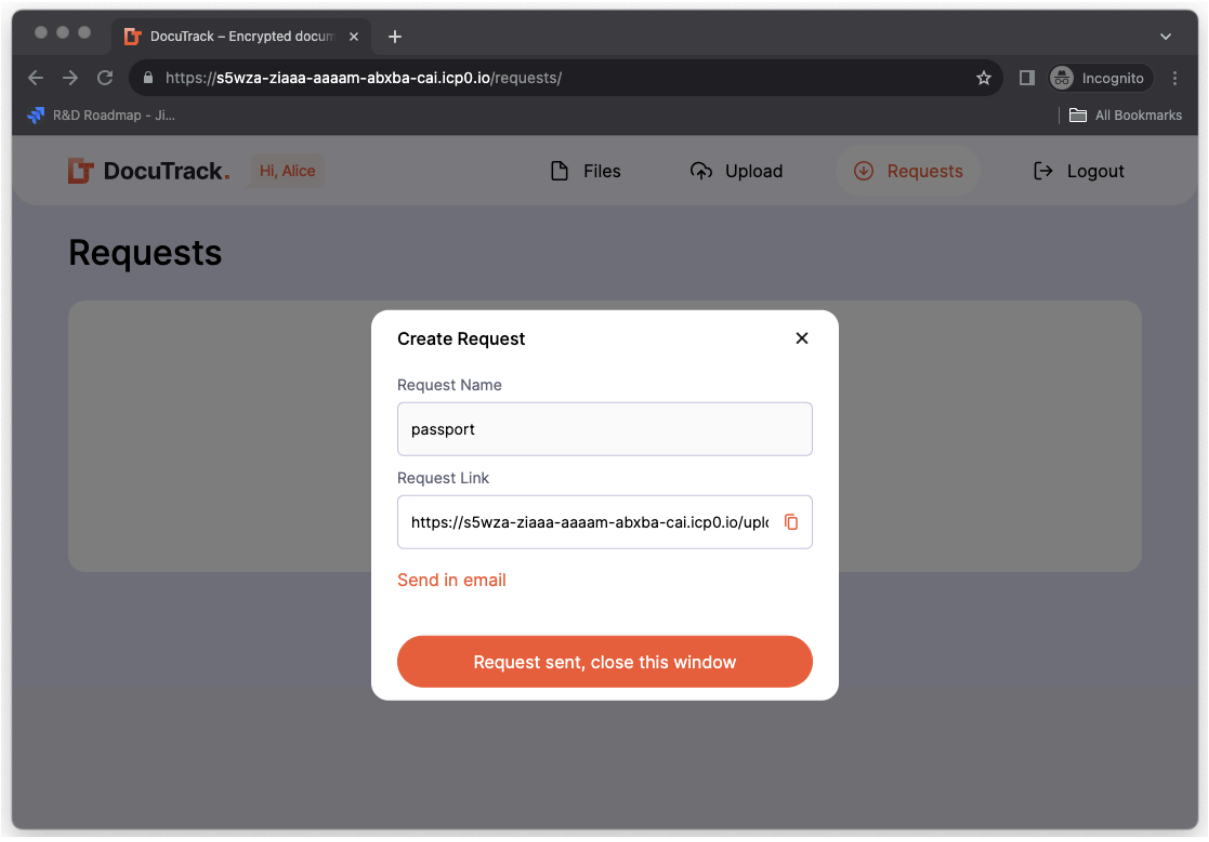
Alice clicks on Login and uses her Internet Identity to authenticate herself to the dapp.



When logging in for the first time, Alice will see an empty list of files she has access to. Later it will show files she's requested from others and who she has shared them with.



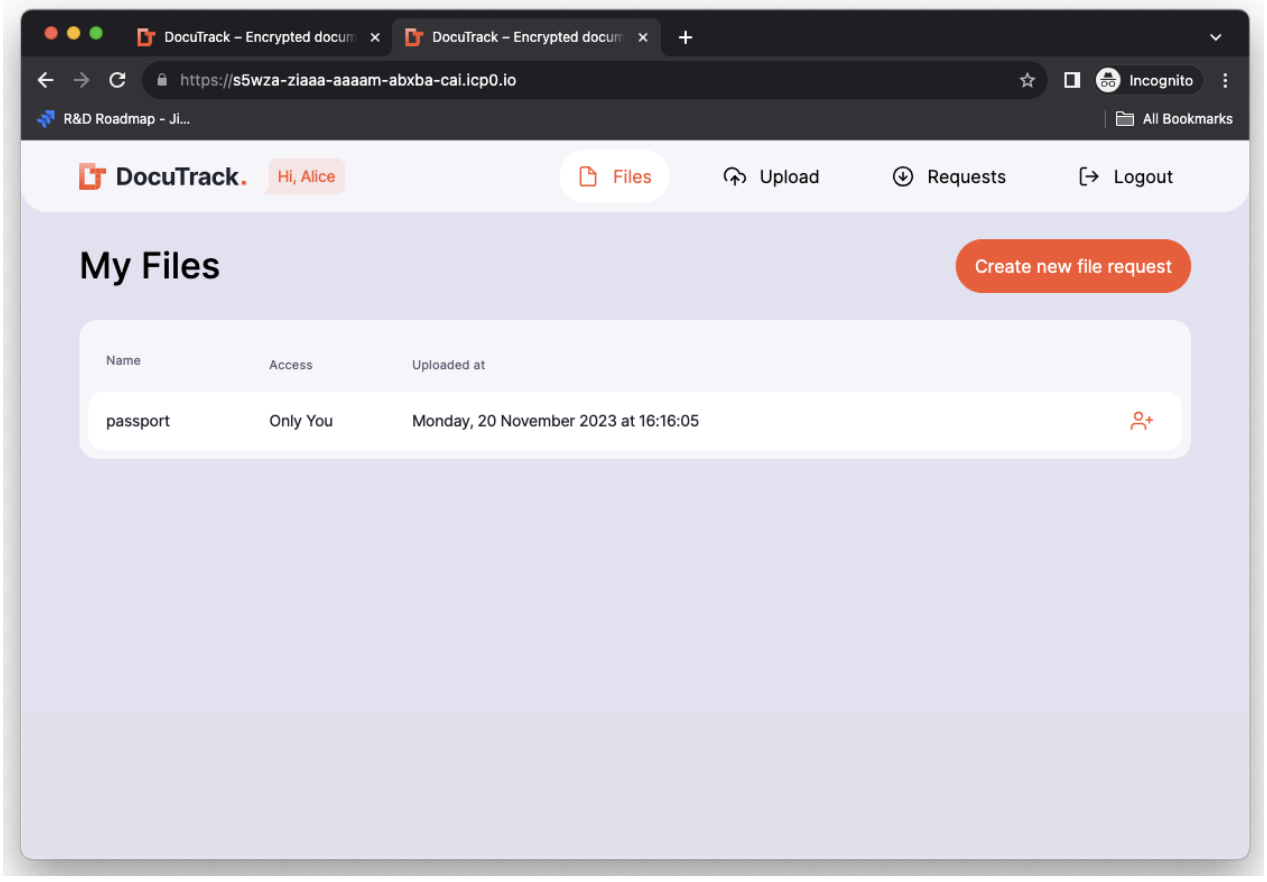
To request a document to be uploaded from Bob (who doesn’t need to have an Internet Identity), Alice can click on Requests on the top right and on the page that opens she can then create a request.



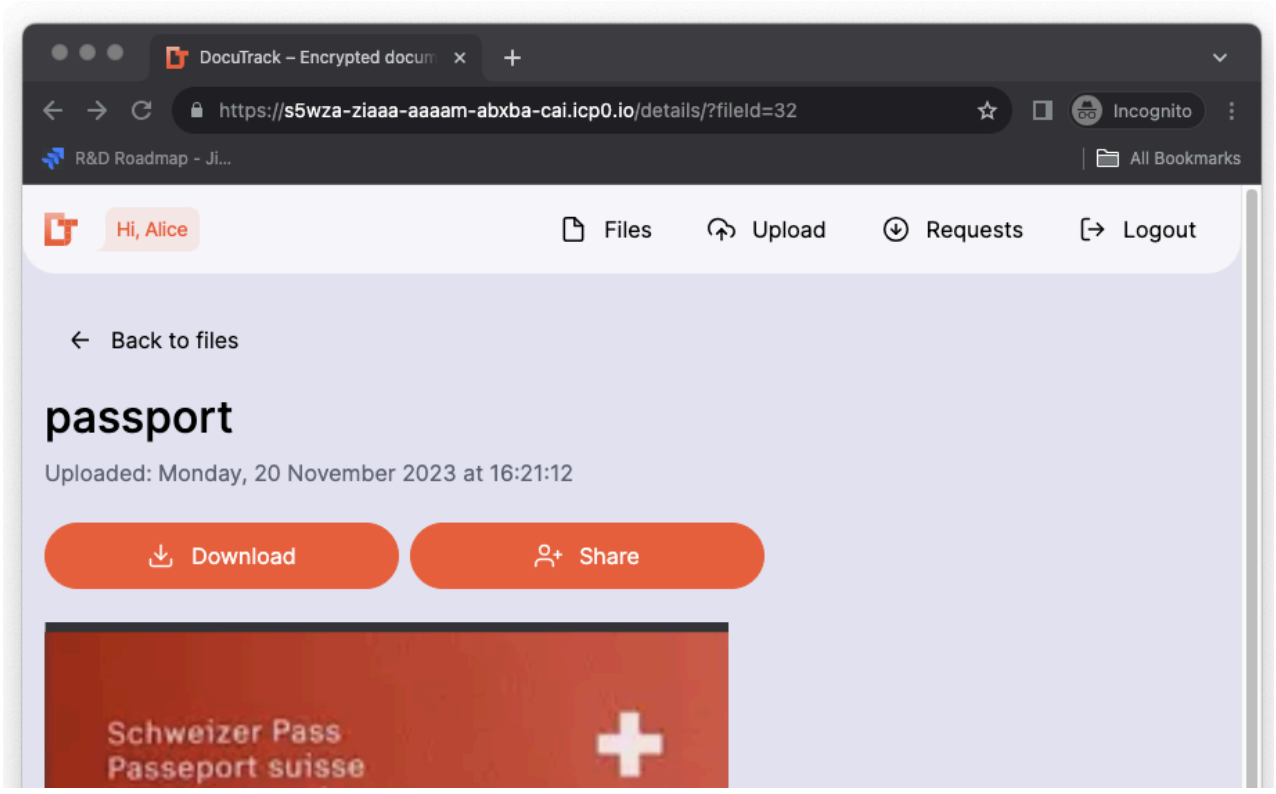
Alice can now enter a description of the file she’s interested in and then send the request link to Bob by her favorite communication channel (email, messenger, …).

When Bob receives the request link and opens it, he can upload a file. Bob does not need to login to be able to do this. After the first upload, the link becomes inactive.

After he has done that, Alice will see the uploaded document under “My Files”.



When clicking on the name, the she will see a file preview for supported document types and she can click download to process the file further.



By clicking on share. She can select people who will also get access to the file and she can set an expiration date for the access.