Nano

Release 1.2.0

Clarabot Zrt.

Apr 21, 2023

CONTENTS

For first time users we highly recommend reading the *Getting Started* chapter of the documentation, which contains all the information and resources you need for quickly setting up a Nano client and corresponding user account.

CHAPTER

ONE

GETTING STARTED

Clarabot Nano is a software pair (Client and web application) that facilitates live file sharing with top security and ease of use.

When it is installed on a user's computer, it will provide controlled remote access to its resources. The permissions and properties of remotely accessible resources are exclusively determined by the owner of the account the Nano Client is using. All user content is secured by end-to-end authenticated encryption between the Nano client and web applications.

The servers used for relaying the requests and responses can't read or manipulate them. The encryption is used in a way to achieve top privacy while also allowing the utilization of central servers. This architecture provides similar data confidentiality to peer-to-peer networks, while also having some of the benefits of a centralized system. Central response caching and broadcasting improves performance, while the network security for the machines running the Nano client is improved by hiding their identity from the connected web applications.

1.1 Foreword on Security

We understand the implications of having a resource sharing software like the Nano client installed on your local computer, with the integrity of your filesystem being at risk.

We put strong effort in eliminating any kind of potential attack vector targeting your machine. All the default settings in the Nano client configuration are set to be secure with as much convenience as possible.

There are several configurations to be made when setting up Nano which are equally critically important to get right for security reasons.

We highly recommend new users to read through the *Webapp Guide* section where we will go over the proper steps to register and use your user account. You'll have a secure account ready to go within a few short steps. With this account, you'll be able to connect to other people's rooms, and by extension to their shared drives as well.

Preparing your own filesystem to work with the Nano file sharing system requires a few additional steps. The *Client Guide* section will help you in setting up the Nano client on your local machine, preparing your computer for accessing your files remotely.

If you wish to know more about our security handling, please refer to the *Security Overview* section.

Some versions of the Nano client will be CC certified. See the *Installation* for more information.

1.2 Terminology Overview

The Nano file sharing ecosystem consists of two software, the Nano client and Nano web application, plus the Clarabot server center. These software work simultaneously together in order to provide the user a fast and reliable file sharing platform.

The Nano documentation will distinctively use the terminology explained in this article when talking about a specific part of the Nano system.

1.2.1 Nano Client

The Nano client software is installed on the user's machine. After the initial setup, the Nano client connects to the server center and allows it's owner to remotely access their shared files. The root folder of the shared files is called a *Drive*; Nano acts as a hybrid file server for the contents of this root folder. The Nano client itself never listens on the network for connections in the traditional sense, it only accepts incoming requests from the server it has connected to. This method of client-server communication promotes better web security than a traditional file server.

Only the owner and explicitly authorized collaborators are able to issue requests to the Nano client. Even the Clarabot servers are incapable of forging and sending false commands to the Nano clients. Even in case of a full datacenter security breach, the users' files will remain secure. See *Security Overview* for details.

Once a Nano client is set up it is available for the owner to be managed remotely through the Nano web application. The extent of the remote management is configurable. See *Client Guide* for details.

1.2.2 Nano web application

The Nano web application is a feature-rich modern website for controlling connected Nano Clients. The webapp also provides several utility tools for teams and individuals alike.

To work together with other Nano users, you must create a *room*. Rooms are customizable groups of participants which you can freely create and edit. These rooms serve a utility purpose for collaborative tasks. By default rooms have a basic message board for group chat services. The messages will be stored on the Clarabot servers.

With the webapp you can create drives on the machine where your Nano Client is running.

You can attach an existing drive to your room which will grant you access to the files shared by your Nano client. The contents of the drive will be available to everyone with membership access to your room.

You can invite other Nano users to your room, giving them membership roles or higher level roles like moderator or administrator.

The Nano web application lists all of your accessible collaboration tools in a workspace. This workspace can be found on the left side of the webapp. The tools within the workspace can be:

- A room, either owned by you, or a room where you have membership access
- A dialogue, which is a direct chat with another Nano user

You can create custom groupings for these tools to create a better workspace overview.

1.2.3 Clarabot Servers

The Nano clients and Nano web applications do not communicate on a peer-to-peer basis, but rather through centralized Clarabot servers. The centralized servers provide fast and secure communication channels for all user commands and data transfer operations.

The server implementation has multiple benefits: Faster communication between the user and Nano client, tighter network security and improved owner privacy.

The servers work on a Zero Knowledge encryption principle. All confidential data sent to the Clarabot servers are encrypted in a way that the servers have no way of decrypting and processing them. You can read more about our use of the Zero Knowledge protocol in the *Security Overview - Goal* article.

1.3 Registration

For details on the cryptographic implementation see *Security implementation in the registration process*

In order to connect to the resource sharing hubs of this system, you must have an account registered. This part of the documentation will guide you through the registration process.

To create a new Nano user account, please navigate to the Nano webapp and click on the *Create an account* link to begin your registration process.

Enter your email and password for your new Nano user account. Make sure you know the credentials for your email. This email will be kept private within the Nano system, but it will be used by the Nano system to send you authentication emails for specific user actions, especially when Two-Factor Authentication (2FA) is set up.

1.3.1 Password strength importance

A collaborative software like Nano must be as secure as possible in order to protect the user's filesystem integrity. The user is responsible for choosing a strong password for their Nano account.

With a weak password, any adversary will have an easy way to either guess it or in a relatively short amount of time, by trial and error, brute force their way into the user's account. To protect the user from a brute-force attack, the Nano servers are set up with rate-limiters which automatically shut down any repeating online brute-force attempts.

In order to alleviate the burden from the user, Nano will help in choosing a strong password at the time of registration.

1.3.2 Password policy

The password policy in Nano must be strict in order to reduce the risk of a data breach.

Nano will give real-time feedback to the user about the typed password's strength in order to help the user in choosing a strong password. This strength check will be done with the zxcvbn library.

CHAPTER

TWO

NANO FEATURES

Before talking about the two main software of the Nano file sharing ecosystem, we'll introduce the main features of Nano.

The features explained in this chapter are implemented in both the Nano Client and Nano Webapp software.

The Nano Webapp is created for the day-to-day usage of these features. Users can freely create Nano accounts and join *Rooms* and explore their connected *Drives*.

The Nano Client software's job is to run on your local computer, handling file sharing requests. Folders that are set up for remote access are shared through cryptographically secured channels. Each and every access of these Drives are thoroughly supervised by your Nano Client.

2.1 Drive

The first step in sharing files through the Nano client is to designate a *drive*. A drive is a folder on which the Nano client has permission to do it's resource sharing capabilities and other supported operations.

The Nano client can do different kind of operations on the assigned drive like listing, searching, creating, deleting, moving, copying or updating files or folders.

After creating the drive on your local machine, you'll be able to assign this drive to a *Room* on the Nano web application.

The drive folder acts as a container for filesystem related operations. Any sort of operation outside this drive folder is not permitted for the Nano client. The Nano client must not and will not know about anything outside the drive's root folder. Any type of references (shortcuts, symlinks etc.) pointing outside the drive folder structure won't work; they'll appear broken.

By default Nano will deny creating drives from directories that may be sensitive to the system or the user. The general recommendation is to only share custom folders that are created by the user, or local drives other than the `C:/` or the POSIX root. This simple rule will mostly solve the inherent security problems stemming from awkwardly set-up share-locations.

Hint: The sensitive path restriction can be toggled off by in the local configuration. See `restrict_sensitive_path_access` in the config.

2.2 Room

Rooms are your personal collaboration tools on the Nano web application. You can freely create new rooms on the fly for different kind of purposes. You can invite other Nano users to your room, allowing them access to the contents within.

A room always comes with a basic chat feature and can have an optional drive attached.

2.2.1 Attaching a drive

You can attach an existing drive to a room. Attaching a drive to a room will let you browse the contents of that drive remotely from your browser.

To keep your workspace simple only one drive may be attached to a room. We encourage the creation of a new room to each drive.

A single drive can still be attached to more than one room allowing multiple groups to access the same resource.

2.2.2 Chat

You can use rooms as a group chat with other Nano users. A room always comes with a basic chat feature.

You as an owner have full membership control over your room. You can also give moderator or administrator privileges to other Nano users to facilitate easier content moderation in larger rooms.

The room chat can function as a message board by setting the respective room configuration. Only users with owner or moderator privileges can create new posts within the room when the chat is set to function as a message board.

2.2.3 Anonymous users

A room can be configured to allow anonymous access. When enabled, collaborators without a Nano user account may enter the room using a room-specific link. The room share link gives anonymous collaborators read-only access to your room, allowing them to read the room chat and browse the contents of the connected drive.

> **Danger:** Always check for sensitive data within your Drive before allowing anonymous user access.

2.2.4 Room management

The Nano web application allows you to make several changes to a room's configuration.

- Allow anonymous access through a specific room share link
- Configure content editing and content sharing privilages
- Give or take administrator and moderator roles to room members

Please refer to the *Nano client remote configuration* for more information about configuring rooms.

2.3 Search

Searching is done in the Nano web application using the searchbox at the top of the page. This search is contextual. A list of possible resource groups will be presented to you as you type in your search terms. These resource groups depend on where you currently are in the web application.

The search results will contain detailed information about each found result and the reason why it satisfies the search terms.

The web application searches within all of your owned Rooms, including all Rooms which are shared to you by other Nano users.

When searching for files, the search is done in the file name and their content as well.

CHAPTER

THREE

WEBAPP GUIDE

This chapter of the documentation will show you how to use the Nano Web Application to it's full potential, touching subjects like:

- Account creation and management
- Drive management
- Room management
- Nano Client management
- Nano license overview
- Using the search functionality

3.1 Account

The Nano account is used to authenticate the user within the Nano file sharing ecosystem.

3.1.1 Overview

A Nano account allows you to securely connect to your configured Nano clients.

3.1.2 Registration

Registering a Nano account is a quick process free of charge. To register, please refer to the *Registration* article.

Make sure to use an active email account when registering. The Nano Webapp will send a confirmation email during the registration process.

After clicking on the confirmation link, your account will be ready for use.

3.1.3 Password strength

Your user account is the main keyholder within the cryptographically secured Nano system. The Nano systems protect your account as best as possible, but you must choose a strong password for your user account so Nano's protection works efficiently.

At the time of registration, or while changing your password, the web application gives you detailed feedback on the currently typed in password's strength. We ask you to heed it's warnings in case any show up.

3.1.4 Account settings

Your account data consists of your chosen email, password, username, avatar, and Two-Factor Authentication (2FA) setting.

Your email is never shown to other users, it always stays private. It's important to choose an active email account for your Nano user account. Nano uses this email address to send client update reminders or two-factor authentication codes whenever they are needed.

Your username and avatar are public. You can use these options to personalize your account. Your username and avatar image are displayed to other Nano users within the rooms you are participating.

3.1.5 2FA

Two-Factor Authentication can be enabled for your Nano account for an extra layer of protection.

When enabled, you will be prompted to enter a numeric passcode before each account-critical request. This ensures an extra layer of security against malicious user data changes.

3.2 Drive

Important: You need an active Nano client for all Drive operations. For setting up a Nano client, please make sure to check out the *Client Guide* chapter.

3.2.1 Overview

In Nano's file sharing system, a Drive is a representation of a local folder on your computer. You can create as many Drives as you want.

It's a good rule-of-thumb to never make a Drive out of your root system folder, `C:/` folder or user folder. You always have to consider that these folders would be accessible to other Nano users if this Drive were attached to a Room with multiple users.

An existing account is required on the Nano Webapp for each drive management task.

3.2.2 Create Drive

On the Nano Webapp, navigate to the Nano manager page by opening the user menu (click on your avatar), then click "Nano manager".

This page will list all you active Nano clients on the right side. Select the Nano client which runs on the machine you wish to create a Drive on (enter *admin password* if prompted).

When a Nano client is selected, click on the "Create Drive" command text which will open the New Drive dialog window.

The New Drive dialog window shows your root system folder structure. Navigate to the folder you wish to create a Drive from. When you open the desired folder, click on the "Select folder" button.

If successful, you will see the newly created Drive in the "Drives" list.

> **Attention:** Operating system specific data folders may appear in the folder list. These folders are unselectable. Nano will give you an alert notification when trying to enter such a folder. If this happens, please select a different folder.

3.2.3 Delete Drive

On the Nano manager page of the Nano Webapp, select the Nano client which hosts the unwanted Drive.

When the Nano client loads, you will see all the hosted Drives listed in the Drives section.

By clicking on the Bin icon, the Drive will be deleted.

Hint: Deleting a Drive will keep the local folder intact, the origin folder will not be deleted from the local filesystem.

3.2.4 Attach to Room

On the Nano manager page of the Nano Webapp, select the Nano client which hosts the Drive you want to attach to an existing Room.

Click on the "Attach to Room" command under the Drive.

A new dialog will open with all the Rooms you can attach the Drive to. When selecting a Room, the Nano Webapp will show you a current snapshot of the Room's membership status.

> **Attention:** Make sure you thoroughly review the Room's membership status before attaching your Drive to a Room. Depending on the Room configuration, every Room member will have access to all your files within the Drive.

3.2.5 Detach from Room

On the Nano manager page of the Nano Webapp, select the Nano client which hosts the Drive you want to detach from a Room.

The Drives list shows you all your existing Drives. Each Drive has it's own sublist, showing all the Rooms the Drive is attached to.

By clicking on the Bin icon beside the Room's name, the Drive will be detached from that Room.

Detaching doesn't delete the Drive, it just separates the Drive from the Room.

3.2.6 Browse Drive

To browse a Drive's contents on the Nano Webapp you have to first attach it to a Room. Please see above how to do so.

Select the Room which has the Drive attached. You can do so by searching for the specific Room on the workspace; the left side menubar on the Nano webapp.

After selecting the Room, switch over to the Drive tab by clicking the Drive button.

The Webapp will list the contents of the Drive.

In this view you can download selected files or upload new ones to the Drive's folder on the local machine.

3.2.7 Search Drive

You can issue search requests to find specific contents within a Drive. To do so you must first select a Drive.

After selecting a Drive, click into the search field on the top of the page and begin typing your search criteria. The Nano Webapp will prompt whether you want to search globally, or within the Room. Searching within the Room also includes the attached Drive.

3.3 Room

3.3.1 Overview

Rooms in the Nano file sharing ecosystem represent a kind of resource group for collaborative tasks. Rooms come with full membership control and a basic chat feature. Optionally a Drive can be attached to the Room, making it's contents accessible for all members.

Without a Drive attached to the Room, the Room functions as a basic message board for it's members.

The owner can attach a Drive to the Room which upgrades the Room with several features. The owner will be able to promote chosen members to administrators, who will be able to help the owner in managing an overly populated Room.

With a Drive attached, several options become available for customizing the Room.

3.3.2 Create Room

To create a Room you have to open the user menu by clicking on your avatar at the top left corner of the Nano webapp.

In the user menu, click the *Create Room*, which brings up the New Room dialog window. Enter the desired Room name and click Save, which will create your new Room.

3.3.3 Room Options

To modify an existing Room you have to first select the Room by clicking on it's name in the Workspace, then click on the Gear icon beside the highlighted Chat tab. This brings up the Room's settings page.

There are several options available for customizing a room:

- Change the Room's name and avatar image
- Enable or disable anonymous access
- Enable or disable message board functionality

Additionally if you are the owner of the room or received administrator or moderator roles from the owner, you can edit the room membership permissions for each Room member, or invite new members using the *invite* button.

You can leave the Room by clicking on the *Leave* button.

Hint: Room administrators only have member management abilities when there is a Drive attached to the Room

---

When *anonymous access* is enabled, the Room becomes accessible to anyone who obtains the Room's *share link*.

When the *message board* functionality is enabled, only the owner and administrators will be able to make new posts in chat.

3.3.4 Room Membership

The Room's settings page shows a table with the current Room members and their membership roles. Each ticked checkbox shows what kind of roles the member has received.

You can modify each members' role by ticking or unticking the corresponding checkboxes.

3.3.5 Delete Room

Deleting the Room won't delete the attached Drive.

3.3.6 Attach Drive

Attaching a Drive to the Room can be done in the *Nano Manager* page. Please refer to the *Drive* article for attaching a Drive to your Room.

3.3.7 Browse Drive Content

If a Drive is attached to a Room, you can browse it's contents within the Room.

By clicking on the *Drive* tab you switch over to the Drive browser view, which will list the Drive's root contents.

In this view you can create new folders on the Drive by clicking on the *New Folder* icon on the top right corner of the page. This action creates the folder on the Nano Client's host machine.

By clicking the *Upload* icon you can choose a file on your current machine to upload to the Drive.

3.3.8 Workspace

At the right side of the Nano Webapp, after logging in, you can find your *workspace*.

The workspace sidebar allows you to quickly overlook and organize the Rooms that are available to you. You can freely arrange your Rooms into custom groups for a better overview. These groups exist only for you.

By default new Rooms are found in the *Ungrouped* group.

You can create new groups by either clicking on the gear icons beside each group name and selecting *Create Group*, or by opening the user menu and selecting *Create group*.

3.4 Client

The Nano webapp gives you the ability to check on all of your active Nano clients.

To open the Nano client manager, open your user menu by clicking your username on the top right corner of the webapp. In the user menu select the *Nano manager* menupoint, which will bring up the Nano manager page listing your active Nano clients.

3.4.1 Overview

The Nano client is a complex multi-process application implemented to be lightweight enough to run in the background without bogging down the operating system. The Nano client is responsible for handling and executing all file-sharing and indexing operations your local machine.

The Nano client is also the main security entrypoint for authorizing all remote file access requests.

For a more in-depth look into the Nano client please refer to the *Client Guide* chapter of the documentation.

3.4.2 Client setup

Please follow the steps explained in the *Installation* article.

3.4.3 Client options

In the user menu select the *Nano manager* menupoint, which will bring up the Nano manager page listing your active Nano clients.

Select your Nano client and enter the admin password if prompted.

With your Nano client selected, the webapp will show you:

- The Nano client's name
- Anonymous access status
- Configured Drives and the Rooms they're attached to

3.5 License

3.5.1 Overview

You can check your current active Nano licenses in the Nano licenses page.

To open the Nano licenses page, open the user menu by clicking on your username at the top right corner of the Nano webapp, and click on the *Nano licenses* menupoint.

The Nano webapp will show you your active license keys formatted into a table. Each row consists of your Nano license key, how many Nano clients you can operate simultaneously with it, and the date when it expires.

3.5.2 Buying Licenses

Licenses can be bought for you own account directly or as a bundle of tokens for giving them to any account freely.

Directly bought licenses may be set up for recurring purchase for convenience.

When buying a bundle of license tokens keep in mind that the tokens will also have their expiration time that starts at the purchase.

3.6 Search

Use the input at the top of the Nano Webapp to issue searches across Rooms and Drive contents.

3.6.1 Overview

When you start a search query with no rooms opened the webapp will give you the option to search globally. Searching while a room is opened will prompt you if you want to search within the room or if you want to search globally.

The search results will contain detailed information about each found result and the reason why it satisfies the search terms.

When searching globally the web application searches within all of your owned Rooms, including all Rooms which are shared to you by other Nano users.

Searching is done in the file name and their content as well.

3.6.2 Search rules

The basic rules for evaluating search terms are:

- Search terms have no priority based on their order
- All terms are looked for, but it is not required for a result to contain every search term word
- Terms are not case or accent sensitive, but exact matches are valued more
- Results where the search term appears more will have a higher match value

3.6.3 Search fine tuning

You can further narrow down your search results with a few different search operators.

- Prefixing a term with "+" (plus sign) will make it mandatory in every result
- Prefixing a term with "-" (minus sign) will make it prohibited in every result
- Search expressions may be created with binding terms together with "AND", "OR", "NOT" operators and parentheses grouping

3.6.4 Search examples

Let's say we have a large amount of different file resources about cats and dogs and other animals, and we want to search for specific results among them.

To search for dogs and cats, where either is enough to be present in a result:

cat dog

cat OR dog

To search for specific phrases:

"black dog"

"white cat"

To search for dogs and cats, where both have to be in every result:

+cat +dog

cat AND dog

To search for dogs and cats, where cat has to be in every result:

+cat dog

To search for dogs without cats:

-cat dog

!cat dog

-cat +dog

NOT cat AND dog

To search for dogs and cats or birds, where dog and cat are insufficient on their own to be in a result:

(+cat +dog) bird

(cat AND dog) OR bird

CHAPTER

FOUR

CLIENT GUIDE

This chapter of the Nano documentation will mostly focus on the Nano Client. The chapter will also go in-detail about the usage of Drives, Rooms and the search functionality.

The Nano client is the software running on your local machine. To begin sharing files from your local machine, you have to set up a Drive for the client. When a Drive is set up, the Nano client will parse and index each file and folder within the drive folder.

There are several extra features and configuration options available in the Nano client. These configurations allow you to fine tune a Nano client installation to your specific needs.

The default Nano client installation will give you plentiful security with only minimal setting up.

The Nano Client was designed to be highly configurable. Nano provides two configuration files which can be edited to better suit the personal requirements of different users and systems.

4.1 Installation

The Nano Client is available in prebuilt binary distribution packages for major platforms. Most users will wish to use these for convenience. People who wish to install from source can find the guide here as well. Either way, download the files of your choice from the download page.

4.1.1 Common Criteria Certification

Some versions of the Nano client will be CC certified. You can see the certified versions on the download page.

Keep in mind, that only the binary software distributions are covered by the CC certification of any version of Nano.

To check if your Nano is certified, open the "About" window and verify that your version was published with CC certification.

Furthermore, the "Common Criteria" tab in the "About" window gives a summary and lists the relevant configuration that are necessary for CC certificated operation. Overview of the relevant configuration(s):

- *Require explicit peer trust*: This configuration changes the identity verification policy of Nano from trust-on-first-use (TOFU) to verified-only. If it is switched on, no peer may request anything from the Nano from any room unless the owner has verified their identity in the web application.

> **Attention:** The "Identity verification" functions are only available as technology preview in the web application. The experimental "Identity Signature" feature in the "Advanced Settings" must be switched on to access them.

4.1.2 Installing binaries

The prebuilt applications have all of their dependencies bundled. They should work if the OS requirements are met.

Windows

For Windows the prebuilt application is available as an installer executable.

Launching it will prompt the user for elevated permissions, where the identity of the publisher is displayed for verification and the signing certificate can be inspected further. The installer will extract the application into the *Program Files* and create a start-menu shortcut. Creating a desktop and an auto-start on login shortcut can be disabled.

macOS

For macOS there are two *dmg* (virtual disk image) files available. One for Intel (64bit) systems and the other for the new Apple Silicone (M1) systems. Download the image of the application corresponding to your device.

Double-clicking on the downloaded file will verify and mount the distribution image. It will contain the Nano client application called *Clarabot Nano* and a link to your *Applications* folder. Dragging the *Clarabot Nano* to the *Applications* folder copies it to the system conveniently. The *Gatekeeper* will verify and prompt you when running the application for the very first time.

Linux

For Linux we provide an AppImage containing the application. It is built on a fairly EOL distribution so most systems should have no trouble running it.

To verify the AppImage, its checksum file shall be downloaded as well. By using the *sha384sum* utility (or relevant alternate for the checksum file) the integrity of the AppImage can be verified. Example of invoking the utility and successful output:

```
$ sha384sum -c clarabot-nano-1.2.0-amd64.AppImage.sha384
clarabot-nano-1.2.0-amd64.AppImage: OK
```

On recently released systems some compatibility packages may be needed for running it, like *libxcrypt-compat* on Arch Linux.

4.1.3 Installing from source

This guide is applicable to all platforms, but it focuses on Linux. This guide is a simple recommendation to give an overview of the steps necessary, but other customizations and configurations are possible.

Prerequisites

- Python version 3.10
- wxWidgets 3+ (for compiling wxPython)
- protobuf library (for compiling gcld3)
- dbus library (for compiling dbus-python)
- OpenJDK (Java) 11

Installation steps

1. Download the latest source package of *clarabot-nano* and *clarabot-crypto* from the downloads page
2. Create virtualenv for nano installation

   ```
   python3.10 -m venv ~/.nano && source ~/.nano/bin/activate
   ```

3. Install the *wheel* package before any other

   ```
   pip install wheel
   ```

4. Install *clarabot-crypto* and *clarabot-nano* packages

   ```
   pip install clarabot-crypto-X.X.X-src.zip clarabot-crypto-X.X.X-src.zip
   ```

5. Launch *clarabot_nano*. It will stop with an error at this point, but it will bootstrap the local configuration. Close it

   ```
   python -m clarabot_nano
   ```

6. Download Apache Tika 2.6.0 and Apache Solr 8.11.2. These version are what get bundled with binary distribution currently. Newer updates to these major releases may work as well.
7. Create directories for solr and tika

   ```
   mkdir ~/.nano/tika ~/.nano/solr
   ```

8. Copy *tika-server-standard-X.X.X.jar* to *~/.nano/tika/*
9. Extract *solr-X.X.X.tgz* and copy the contents of the extracted *solr-X.X.X* directory to *~/.nano/solr/*.

   ```
   cd ~/.nano/solr && tar --strip-components=1 -xf solr-X.X.X.tgz
   ```

10. Edit `~/.config/clarabot/nano/local.yml` by uncommenting and setting `solr_bundle_dir` to `/home/YOUR_USERNAME/.nano/solr` and `tika_bundle_dir` to `/home/YOUR_USERNAME/.nano/tika`. They are at the bottom of the file.
11. Start nano with

    ```
    python -m clarabot_nano
    ```

    OR

    Create script to start nano by saving the following to a file, set execute permissions with `chmod +x filename`, then double-click on the file to run nano

    ```
    #!/bin/bash
    source ~/.nano/bin/activate
    python -m clarabot_nano
    ```

4.1.4 After installation

After starting the application, log in with your Nano user account credentials. If you don't have a Nano account set up, make sure to do so by following the steps explained in the *Registration* chapter.

The Nano Client comes with a default configuration wizard. The wizard will help you in setting up your client for first time general usage.

4.2 Drive

4.2.1 Overview

In Nano's file sharing system, a Drive is a representation of a local folder on your computer. You can create as many Drives as you want.

4.2.2 Create Drive

Hint: This process is best done on the Nano webapp

Creating a Drive requires a Nano user account. To create an account, please read the *Registration* article.

To create a Drive using the Nano client, first start your downloaded Nano client software. The software will prompt you to enter your Nano user account credentials.

After logging in, navigate to *Nano -> Configure -> Drive manager*, which will open the Drive configuration manager window.

In the Drive configuration manager window, you will see a list of your currently set-up Drive folders. If it's your first time setting up a Drive, this list will be empty.

Click on the *Create drive* button, and choose a folder you wish to create a Drive from.

After choosing, you will see the path of the chosen folder appearing in the Drive configuration manager list.

It's a good rule-of-thumb to never make a Drive out of your root system folder, `C:/` folder or user folder. You always have to consider that these folders would be accessible to other Nano users if this Drive were attached to a Room with multiple users.

4.2.3 Delete Drive

Hint: This process is best done on the Nano webapp

To delete a Drive navigate to *Nano -> Configure -> Drive manager*, which will open the Drive configuration manager window.

In the Drive configuration manager window, you will see a list of your currently set-up Drive folders.

Right-click on the Drive you wish to delete, and select *Delete drive*. This will remove the Drive from the Nano file sharing manager.

Hint: Deleting a Drive will keep the local folder intact, the origin folder will not be deleted from the local filesystem.

4.2.4 Attach to Room

Attaching a room (server entity) to a drive (local/Nano entity) securely requires the Owner to provide Nano the current permission configuration of the room. This ensures that the Nano will only serve any room, starting from the state the Owner has verified. The web application conveniently presents this process, but it can be performed on the client as well. Attaching rooms using the UI of the Nano client is primarily necessary when the `remote_admin_policy` is set to `deny` and it can't be performed from the web application.

To attach a Room to a Drive navigate to *Nano -> Configure -> Drive manager*, which will open the Drive configuration manager window.

Right click on the Drive you want to attach the Room to and select "Attach room to drive". Follow the instructions in the window that comes up.

4.3 Configuring local Nano Client

The `local.yml` configuration is the base of a running Nano which handles the Nano instance, dependencies and localhost security.

For security reasons, Nano can not update this configuration file. To ensure that this rule won't be circumvented it is recommended to set this file to read-only.

Updating this file requires the Nano to be restarted in order to see the changes.

4.3.1 `log_config`

Nano uses the builtin Python logging module for it's internal logging tasks. For configuring the logging module, please refer to the official Python docs .

4.3.2 `remote_admin_policy`

default: `"restrict"`

This option sets the way Nano handles the remote configuration commands issued by the Nano owner.

Three settings are accepted here:

- `allow` accepts all admin commands
- `restrict` will prompt the owner to input a passcode of 5 digits along with each command
- `deny` rejects all admin commands

The remote configuration commands update the settings found in the `remote.yml` configuration file.

Enabling this security policy may greatly limit the damage an adversary could do by gaining access to the account of the owner. The downside however is that even the owner will be unable to change configurations remotely, which could be highly inconvenient depending on the location of the hardware which runs the Nano client.

4.3.3 localhost_network_security

default: "ec"

This configuration determines how the different processes in a nano instance authenticate and communicate with each other. By default Nano ensures authentication and confidentiality during process communication. This can be changed to only require basic authentication which consumes less system resources.

Two settings are accepted here:

- ec authenticates processes and encrypts communication with dynamically generated x5519 curves
- basic authenticates processes with dynamically generated plain passwords

4.3.4 path_unicode_normalization

default: "default"

This configuration determines how paths from remote systems are normalized before writing to storage.

Valid settings accepted here:

- default will be NFD on Mac and NFC on other platforms
- keep will not normalize, just keep the remote paths as-is
- NFC, NFD, NFKC, NFKD can be selected directly to be used

4.3.5 restrict_sensitive_path_access

default: true

Deny creating and serving drive roots of common places with sensitive data.

Danger: The configurations below handle software dependency - incorrect setup will stop Nano from booting up

4.3.6 config_dir

default: ""

An absolute path pointing to a writable directory where necessary configurations will be stored, such as drive, authentication etc.

4.3.7 data_dir

default: ""

An absolute path pointing to a writable directory where necessary data will be stored like sessions, state, etc.

4.3.8 pid_dir

default: "pid"

An absolute or relative path pointing to a writable directory where subprocess tracking information will be stored. This directory must be on a local filesystem.

Note: If the pid_dir is relative, it will be appended to data_dir

---

4.3.9 solr_dir

default: ""

An absolute or relative path pointing to a writable directory where Apache Solr stores index data and configurations. These paths can be overridden by more specific configurations (see below).

The directory should be on the local filesystem. (NFS is theoretically possible, but not supported well)

This location should be secured as properly as the data itself from local access (other users).

4.3.10 solr_data_dir

default: "data"

An absolute or relative path pointing to a writable directory where Apache Solr data files will be stored.

Note: If the solr_data_dir is relative, it will be appended to solr_dir

---

4.3.11 solr_logs_dir

default: "data"

An absolute or relative path pointing to a writable directory where Apache Solr log files will be stored.

Note: If the solr_logs_dir is relative, it will be appended to solr_dir

---

4.3.12 tika_dir

default: ""

An absolute or relative path pointing to a writable directory where Apache Tika stores log files and configurations. This path can be overridden by more specific configurations (see below).

The directory should be on the local filesystem. (NFS is theoretically possible, but not supported well)

This location should be secured as properly as the data itself from local access (other users).

4.3.13 tika_logs_dir

default: "logs"

An absolute or relative path pointing to a writable directory where Apache Tika log files will be stored.

Note: If the `tika_logs_dir` is relative, it will be appended to `tika_dir`

4.3.14 java_dir

default: ""

An absolute path pointing to a directory where JRE is located.

Some dependency software which Nano uses are Java based and require a JVM to run.

If left empty, Nano will try to lookup the JRE from default locations:

- Use JRE that's bundled with the client
- Use JRE indicated by JAVA_HOME environment variable
- Use JRE found in PATH (java executable)

4.3.15 solr_bundle_dir

default: “”

An absolute path pointing to a directory where Apache Solr is located.

If left empty, Nano will try to use the client's bundled software.

4.3.16 tika_bundle_dir

default: “”

An absolute path pointing to a directory where Apache Tika is located.

If left empty, Nano will try to use the client's bundled software.

4.4 Configuring remote Nano Client

The `remote.yml` contains the properties of Nano, like served resources, resource permissions, authenticity, indexing options and more. When the owner changes a configuration locally by the UI or remotely from the browser this file is updated.

This file can be updated remotely as well by the Nano API for the convenience of the owner. Because of the security risks coming with this convenience, the owner is highly encouraged to consider their security needs and restrict editing this file by either:

- Set `remote_admin_policy` in the `local.yml` to either `restrict` or `deny`
- Make this file read-only for the Nano process

Depending on the choice, a high degree of security will come with some inconvenience. If this file is read-only or it's editing is disabled by policy, the owner will have to revert these settings to make changes to the configurations in this file through the Nano API. This either requires physical access by the owner or the use of a secure remote management system such as SSH, RPD, etc.

4.4.1 name

default: "" (*hostname*)

Name of the Nano instance for convenience. By default this will be the hostname of the machine when a new configuration is generated.

4.4.2 admin_password

default: null

The admin password is only used if the `remote_admin_policy` in `local.yml` is set to `restrict`.

Do **not** set this as a plain-text password! That is insecure and will not work. The password value must be set through the software to be hashed for secure storage.

4.4.3 drives

default: []

Example configuration:

```
drives:
- path: /mnt/data/documents
  rooms:
    - 4CM7XP733333
    - 4CM7YVQ33333
```

path: Local path in the filesystem that shall be the root for the drive
rooms: List of the rooms by their id that the drive should be attached to

The owner of the Nano can declare different system resources to be handled (that the Nano supports, for example drives). These drives need to be mapped to a room so the server can make it available to the desired audience.

For more freedom Nano allows mapping a drive to multiple resources. This is convenient in case different groups should have different default permissions on the drive, or the groups should not know about each other.

If the path of a drive becomes inaccessible, searches will still work for contents already indexed, but no other operations will succeed.

> **Warning:** If multiple resources get the same room listed among different Nanos of the owner then those rooms will be in a competing-conflict state for the given resources. Nano will notify the owner if such a conflict state occurs.

4.4.4 room_blocks

default: []

Example configuration:

```
room_blocks:
  - 4CM7XP733333: ["2467z86822222:0GLPOdcixTh3W0e/j9TUfEPaXl398pvTvN/w12miXS8="]
```

These values are automatically updated to the latest known blocks by the Nano because verifying the authenticity of new blocks is secured by the owner account's keyring with digital signatures.

All membership permissions are stored on the server secured by a digitally signed blockchain which only the owner is ever able to edit. The only malicious action a server would be able to do is to drop the top N block of the chain (lie of omission). The Nano can prevent even that by storing and requiring that the latest known block of the chain is always available. If the stored identifiers of a room are not present or seem to be invalid, the owner must re-attach the room to the Nano.

4.4.5 deny_anonymous

default: false

This setting provides a local policy override over the permission settings of rooms.

Permissions of a room's config blockchain may indicate that the associated resources should be accessible by anonymous requests. For extra security the owner may set this to true, which will block any anonymous access from any room.

4.4.6 require_explicit_peer_trust

default: false

This setting provides a local policy enforcement of the trust level for peer accounts to be accepted for communication.

For extra security the owner may set this to true, which will disallow any peer account access unless their identity has been explicitly marked as trusted by the owner.

4.4.7 indexed_languages

default: []

The languages specifically supported for accurate indexing by the search database. Languages that are not specified will be indexed in a generic field with reduced accuracy. The more languages set up the more costly it will become to search.

In most expected cases primarily one or very few languages will be present in the indexed documents, so this is a great place to optimize the search times.

Valid arguments:

- `all`: All supported languages. (not recommended if the quickest search times are important)
- `[]`: The system locale at the time of application and english
- List of ISO language codes: `hu`, `en`, or `de`. Not supported language codes will be ignored. If none are supported then the config will be treated as if empty

Warning: Changing this value will cause all the indexes to be deleted and all content to be re-indexed.

4.4.8 indexer_sync_interval

default: 4.0

4.4.9 indexer_remove_lost_count

default: 3

4.4.10 indexer_force_sync_on_startup

default: true

Operating systems do not offer a system to reliably and robustly track changes on their filesystems. This will never be solved universally due to technical limitations or the performance/resource cost such properties would require.

The indexer process will use filesystem events where they are available, but even in such cases changes can be missed if the nano is not running when those occur. Even on a local filesystem, events from a path may not be generated if they involve symlinks.

Because of these issues a periodic polling synchronization must be used no matter the platform or underlying filesystem. The synchronization must also include checking if an indexed file still exists as their removal event could also be lost, if generated at all. Some network drives could become unavailable for an extended period of time, but might eventually come back again.

To handle such cases and avoid removing and re-indexing files all the time, indexed files will be marked as lost and will only be removed after they have been consecutively missing for a number of synchronizations. In addition to the consecutive missing count the removal will only take place if the first missing state was logged at least `indexer_sync_interval` multiplied by `indexer_remove_lost_count` long ago.

4.4.11 indexer_minimum_delay_between_path_sync

default: 0.0

The indexer processes and re-processes files for searching periodically and by filesystem events. This periodic reprocess work can be excessive at times, so throttling it is made possible here.

Two file syncing of any kind will have to wait this delay between each other at least, making the work more spread out in time. This may be the most useful to tweak when the same files are frequently and rapidly changed again and again.

This value is in seconds and it may be set to 5 at most. (Setting a higher value will default to 5)

4.4.12 indexer_reserved_space

default: 3000

The unit threshold in megabytes for the index storage. If the index storage falls below this threshold, the indexer will halt all processes until enough space is available for work to proceed.

4.4.13 indexer_ignored_mimes

default:
- application/javascript
- text/css

The indexer should not process all file types unconditionally. Many files can be considered very technical and unnecessary for indexing. For example if a user saves a web-page from the browser, many resource files will be saved aside from the HTML. Indexing these would not benefit the user in most circumstances and they would require language specific tokenizers for optimal matching.

4.4.14 indexer_extract_text_limit

default: 1500000 (almost 3 times the size of War and Peace)

Limit of the indexer for extracting text from a resource.

4.4.15 remote_files_restriction

default: true

The servers can attempt to set/use zone-info data to the files that are modified by remote requests. Such info is usually platform, filesystem and OS specific if available at all. They can provide some protection by preventing execution of untrusted files.

4.4.16 server_process_number

default: 1

The number of general server processes can be increased for Nanos that have a very high traffic to improve performance.

4.4.17 secrets

default: []

Encryption secrets are used for various steps. These are automatically generated and may be refreshed periodically. They can be replaced on demand by an admin-request. Modifying them manually is not possible, the last value is a fingerprint of the others and the local machine's identity. If the configs are copied by some deployment system the new machines will automatically generate secure secrets.

4.5 Instancing

Because of it's microservice architecture, Nano is able to run on multiple instances on the same machine by the same or multiple users without issue.

By default Nano will start in the default instance location for all users. This default home directory is different for all operating systems.

Creating a new Nano instance is simple. You just have to create a new *local.yml* configuration file and pass it to Nano in a start argument. As the name suggests, this file is not editable remotely.

Warning: Instances must not mix used paths aside from the dependant software!

4.6 Processes

Under the hood, Nano is separated into different processes by the rules of microservice architecture.

4.6.1 Core

The core's purpose is to handle the different instances and configurations. It also manages the other running processes.

4.6.2 Control

A minimalistic user interface which allows the user to control the running Nano instance.

4.6.3 Indexer

The indexer takes all the configured paths and traverses them in order to index their contents. It's other task is to observe these locations for any content changes.

4.6.4 Gateway

The gateway process is the main entrypoint for all server communications. It behaves as a request preprocessor and request handler for different kinds of actions.

4.6.5 Server

The server process is split into two parts by the type of requests.

FsTx

Filesystem transactor is responsible for executing requests that create or update physical files on the system.

General

The general process handles all listing and get requests on the file system, including administrative and search requests.

4.6.6 Picoture

This is a custom thumbnail generator worker process. It works similarly to the Tika content processor. The indexer uses it to produce thumbnails for supported files.

4.6.7 Additional software processes

Nano uses a couple third party, open source softwares for the raw calculations of the indexes and file contents. These software are Apache Solr and Apache Tika.

Apache Solr

Apache Solr's query engine (in standalone mode) is used for creating the search indexes and maintaining the index database.

Apache Tika

For file content extraction, Apache Tika is used.

4.7 Supervision

4.7.1 Overview

Nano logs important status changes, errors and remote request execution. These can be utilized by the Owner to troubleshoot their Nano and inspect remote access to their machine.

Log configuration including, logging level, file rotation and retention are configurable, see *Configuring local Nano Client* for more info.

Hint: The default location of the log files is different for each platform:

- Windows: *%LOCALAPPDATA%\clarabot\data\nano\logs*
- macOS: *~/Library/Application Support/clarabot/nano/logs*
- Linux: *~/.local/share/clarabot/nano/logs*

4.7.2 Service Log

The service log can be accessed from the UI in the "Logs" menu. This file holds high-level status reports and errors from all processes in the Nano instance.

Inspecting this log should be one of the first things to do when troubleshooting operation anomalies.

Hint: The file is named *process.log* on disk.

4.7.3 Access Log

The service log can be accessed from the UI in the "Logs" menu. This file holds summary information for all remote requests that the Nano receives.

CHAPTER

FIVE

SECURITY OVERVIEW

This chapter of the documentation presents the security goals of the Nano system. We will also touch the used cryptographic implementations within the data transfer protocols. The chapter's aim is to explain how the system provides secure user account management, request handling and client-to-client communications.

5.1 Design goal

The Nano ecosystem is designed with the goal to provide an easy to use data sharing platform with unparalleled security and privacy. This high level of security and privacy is based on the unique way of handling each message which goes through Nano's network.

Nano's network is not peer to peer, but rather a hybrid server-client network.

The Nano system has centralized Clarabot servers which act as proxies for the data sharing network. The Clarabot servers provide robust security for the clients in the network. Without the right permissions, no Nano user is able to snoop into the files of a different Nano user. The hybrid server-client network also makes it possible to cache the separated, encrypted data streams for a shorter request round-trip-time.

Privacy is guaranteed despite the centralized servers. The servers do not store the user's data, they only relay it in a safe and secure manner.

The user stays the owner of their data.

5.1.1 Layered Cryptography

At the time of registration, multiple keys are set-up for the user by the Nano client for different purposes. It's crucial to choose a strong password for the user account. Without a strong password Nano won't be able to provide you with the highest level of security. We encourage every user to read the password policy in the *getting started* chapter.

Using a strong password allows Nano to set up a durable security fortress around the user's private data. The security implementation consists of multiple layers of different cryptography methods. This gives your shared data unparalleled security within Nano's collaborative file sharing system.

For an in-depth tour through the different cryptographic protocols used in Nano's system, please refer to the *cryptography chapter*.

5.1.2 Transparent implementation

The security keys for the user are created client-side on the user's device. This promotes a zero knowledge principle within the Nano system. The sourcecode for this client side implementation is made publicly available for independent verification.

5.1.3 Zero knowledge

The central servers work on a strict Zero Knowledge principle. Zero knowledge means that beside the owner no other user can access the transmitted data. Not even the Clarabot servers have the ability to decrypt the transmitted data. This ultimately removes the risk of userdata exposure in case of a central server breach.

5.1.4 End to End encryption

End to end encryption means that no sensitive data leaves the users' local machine in clear form. Everything goes through cipher processes before it's handed over to the messaging pipeline, only to be decrypted by the receiving end. Because of this process, there is no way to decidedly alter the data being sent without the receiving end noticing.

The end to end encryption combined with the zero knowledge design means that no other parties - not even the servers at Clarabot - are able to access the user's data. This serves as a building block to facilitate secure dialogues between users, and by extension, group of users.

5.1.5 Handling of Unencrypted data

Some limited data of a user account is not encrypted, for example the user's email address. This data is required to be kept unencrypted in order for the servers to provide certain services. The handling of this information is in accordance with the rules laid out in the general data protection regulation (GDPR).

Please refer to the privacy statement on other technical data management like IP addresses, billing addresses, etc.

5.2 Browser security

The browser environment is used for the primary user application because of its unparalleled support on all platforms.

The web-application is served from the Clarabot servers directly. These programs and resources are protected in transit by the standard TLS technology that is widely supported and used. The user's account is managed by the web application received from the Clarabot servers, which means they need to trusted to provide authentic and secure executable files. This is the most convenient use of Nano that requires no installation on the user's side.

5.3 User Account

5.3.1 Registration

The account registration procedure is the cornerstone on which all other cryptography is placed later on. It is described in detail in the *cryptography* chapter.

As an overview, the client machine produces the account keyring and encrypts it in a way that only by knowing the account's password it is possible to decrypt it. The encrypted keyring is stored on the server for the account. Provided the password is sufficiently strong, these data can't be manipulated by the server and their storage is safe.

5.3.2 Login and session

The login procedure does not reveal, if either the email or password is incorrect. This prevents nosy individuals to discover which email addresses are registered in the system.

The login attempts are strictly rate limited, so brute-force guessing login credentials is out of the question.

After a successful login, the client's session keys are stored and used in ways to prevent CSRF and XSS attacks. All private keys of the account are stored uniquely encrypted making remote session destruction possible while also potentially keeping the account's keyring safe.

Both the login procedure and session handling are described in more detail in the *cryptography* chapter.

Attention: The security of the user's machine and its software are up to the owner or their administrator, it is not the responsibility of Clarabot Zrt. Generally speaking, a strong passphrase for the user and utilizing full-disk-encryption are useful.

5.3.3 Account recovery

Losing an account's password may render it permanently inaccessible. The cryptography of the account is required to be operable for all functions by design. Since the server is intentionally locked out of access to the keys by encryption, it is fundamentally impossible to reset an account's password by support as usual.

The password can be changed even if it is already lost, as long as there is still an active login session for the account.

We encourage our users to save a recovery key for their account. It is the only way to recover an account of which the password was lost and no active sessions remain. However, the recovery key contains sensitive information and the user must securely store it. Depending on circumstances it could be stored offline and be put into some password manager or vault either digitally or physically.

If the email account is not accessible, the support can help in changing it. (procedure yet to be defined and implemented)

Account recovery is described in a little more detail in the *cryptography* chapter.

5.3.4 Two factor authentication

This provides additional resilience against a malicious party stealing an account. Several operations that are related to account security are automatically and unconditionally protected by two-factor authentication.

5.4 Peer identity

The peer/public keyring of an account consists of their public keys and signatures of those by their respective signing key(s).

This makes the signing public key usable as a root-of-trust for each account and it ensures consistency within the keyring.

Account keyrings are used for peer verification tasks.

5.4.1 Peer trust

For improving global protection, each peer's cryptographic identity can be fingerprinted uniquely by each account which is their explicit trust mark for the peer. These peer identity fingerprints are stored on the server securely encrypted. Additionally all clients use a trust-on-first-use (TOFU) identity store. TOFU is used to provide consistency for the identity of connected peers where no explicit peer trust is available.

These measures provide protection from man-in-the-middle attacks, or if the integrity of a keyring is compromised by a malicious server.

5.5 Access Control

5.5.1 Single user

By default only the owner of a Nano has exclusive access to the resources being shared. The owner can issue administrative commands that change the Nano's configuration.

Administrative commands will require the owner to provide a password specific to their Nano. This provides some security for the worst case scenario, when the owner's account is effectively stolen by a malicious party.

5.5.2 Multiple user rooms

The only way other Nano clients may request anything from the owner's Nano client is through a room. Rooms are entities on the server that the owner creates. A Nano client can be bound to a room using a drive. Rooms have a configuration stored by the server that contains the membership and group permissions.

Room encryption

Room configurations are stored on the server. On the server, room configurations form a digitally signed DAG (directed acyclic graph) blockchain. This helps in keeping them secure against manipulation. Only the owner is able to make blocks that their Nano will accept.

The server is denied the possibility of omitting the top N blocks because the Nano client will save the hashes of the top blocks. This helps in lie-of-ommission events. If blocks become missing the owner will need to re-validate the room-config and attach it to the Nano's resource.

All rooms have a secret key that is automatically shared with new members. This key is used to encrypt server-powered features. The key is also used as salt for the end-to-end encryption key that the Nano expects for a specific room.

Room configuration

The server cannot manipulate the config of the room, but it can read it. By doing so the server can provide pre-filtering for requests. For example an account that is not authorized to access a room will be denied by the server to send a request. (The Nano client would not accept it anyways.)

Other configuration options will allow you to set whether anonymous access is allowed (by share-link). You can also configure content editing, content sharing, administrative and room access permissions for each individual user.

The message board of a room can be configured to be read only. This way only the owner and the appointed administrators and moderators will be able to create new messages.

Room permissions

The owner has absolute control over the permissions of their room. In case of very populated rooms, the owner may want to allow trusted accounts to help in administrative tasks. Members in a room-config can be appointed the administrator role. This grants them the unique permission to ask the Nano to perform room-config changes in their name.

Anonymous access

By default only the owner and invited, authorized users can access the room. This can be overridden in the room configuration so that anonymous users may also see the contents of the room.

This configuration enables a room specific share-link. This share link contains the room secret key. The enabled link can be used by anonymous collaborators to enter the room and see it's contents unencrypted.

5.6 Message encryption

The message encryption in Nano was determined by 3 factors:

- The server should be able to store the response a relevant fast-access cache storage
- Responding to requests from this cache storage must be as secure as responding from the destination Nano instance
- The server must not respond from cache without consulting the destination Nano instance

5.6.1 End to End protocol

Nano only ever accepts requests in authenticated-encryption, and also responds in a similar manner.

The end-to-end protocol of the Nano is built to help the server to utilize caching of encrypted data chunks. Despite this caching feature, it is still guaranteed that a server won't be able to serve a request without consulting the responding Nano.

This is because a request is always encrypted in a way that only the Nano's owner account can decrypt it. Since it is encrypted the server cannot discern what the request is. The encryption means that the server cannot respond to a request from the cache outright.

5.6.2 Response kinds

Nano may separate the response into encrypted chunks depending on whether the server is able to cache the response or not.

The first chunk is responsible for setting up the security and authenticity of the following message parts. The rest of the message parts are also encrypted and can only be decrypted with the key included in the initial response.

After separating the response into chunks, Nano sends an initial response that holds a request-authenticity-hash. This request-authenticity-hash is built up from the request's metadata. The authenticity-hash provides unfalsifiable proof that the response is for this specific request and the server is not trying to mislead the client.

The initial response also contains the count of the chunks the content will be separated into. With the data received in the initial response, the client can securely request the wanted message. This request will then be served from either the server cache or the Nano client.

Because of this two-step response handling, even if the server can send the response from the cache storage, the initial request is always sent to the Nano client. After authenticating the request, the Nano client then responds with a cache or transfer key that the server can use. These keys are encrypted as well, the server does not know what they represent.

Cacheable response

Cacheable responses are always separated into multiple parts. The initial response will hold cache-key, and a part-key that is the unique encryption key of the whole payload.

The cache key is the best-effort state representation of the data source. In case of a file it includes multiple information from which the file's recentness can be deducted (the inode, size, mtime and path for example). A change in any of those will indicate that the cache-key is stale and the client needs to restart the request. This automatically invalidates previous cache entries. The cache keys are securely encrypted.

Un-cacheable response

Responses that the server is unable to cache are equipped with a transfer key in place of the cache key.

Cache, Transfer and Part keys

Cache/transfer keys and part keys are always derived from strong cryptographic secret values generated and unique to the Nano instance.

5.6.3 Replay Attacks

Requests that are not idempotent, or in other words requests that would result in a mutation/state change on the Nano client require the requester to set-up a session-id. This session-id is used to prevent replay attacks by a malicious server. This session-id (or mutation-id) consists of a session token which identifies the current command request in the context of the current client-Nano connection. The session-id also has a sequence number which the Nano saves in order to challenge the next command with the same session id.

5.7 Server chat

Nano provides two options to help coordinate and communicate between Nano users. These options are a basic peer-chat (dialogue) and a message board type room chat (group chat).

Both types of communication options are securely encrypted by their keys. These keys are private, never used directly. These communication keys are always derived by different factors for an encryption key. So the actual encryption keys change over-time automatically, but their secret input-key-material do not.

5.7.1 Dialogue (P2P chat)

The dialogues are encrypted using the Elliptic-curve Diffie–Hellman (ECDH) key agreement protocol. This allows the two parties to establish a shared secret key using their own account keyrings. This method of key sharing results in a secure shared key even within an insecure channel.

5.7.2 Room chat

A room chat is akin to a message board in functionality. The room message board allows multiple users to communicate with each other in a secure environment. The message board is encrypted with a room secret key. As mentioned in the *room configuration* section, the owner is able to configure a variety of permissions for each user or for the room globally.

One of the configurable options for the room chat is the anonymous access feature. This allows the owner of the room to enable a shareable link. The share link can be used by anonymous collaborators to read the message board and access the contents of the connected Nano drive.

Caution: Because of the inherent security risk of the share link, it is highly advised to only enable this feature for rooms that do not contain confidential data.

If a room is compromised, the proper action to take is to create a new room with members excluding the untrusted ones and deleting the old room.

CHAPTER

SIX

CRYPTOGRAPHY

This is where we talk in-depth about the cryptographic implementations within Nano.

The end-to-end encryption used by Clarabot Nano has several security benefits that many other software lack. However, it is important to understand that cryptography can't provide complete security on its own. Each user must protect their account and devices from being compromised by third parties, be it a person or a computer virus. The security of the user's account and devices are outside of the control and responsibility of Clarabot.

6.1 Cryptography guidelines

All communication is designed with security and privacy in mind. Here we discuss some details that are generally true for the systems.

6.1.1 Versioning

All cryptographic primitives and ciphertexts in the system are versioned in their context. This means seamless replacement of old ciphers or configurations may take place once the new cryptography has widespread support.

6.1.2 Symmetric cryptography

The AES symmetric encryption cipher is used with AES-SIV mode primarily. This mode is one of the most robust that is available, but it's more resource demanding. The performance loss is relatively low because it is a clever construction of two other AES cipher modes: AES-CBC and AES-CTR. These two have support by the WebCrypto API in all relevant browsers. Using the high performance native modes, the performance of the AES-SIV construct is comparable to popular Javascript implementations of alternative ciphers like ChaCha or Salsa.

Noteworthy properties of AES-SIV:

- It is an authenticated encryption, meaning it provides authenticity in addition to confidentiality.
- The security it provides hardly degrades even when accidentally using repeating nonce values.
- It is usable for key-wrapping.
- It is a stream cipher. (even though it's offline, meaning it needs two passes over a data)

Security value choices:

- Nonce values are 12 bytes of CSPRNG or securely derived hash values.
- We use keys that are equivalent to having 32 bytes of security for most AES modes. For AES-SIV this actually requires 64 bytes of length. These are usually generated with HKDF from a 32 byte true key.

Future improvements:

- Once the WebCrypto API is updated, using AES-GCM-SIV would improve performance greatly.

6.1.3 Hashing

Hashing is a multi-purpose cryptographical operation with many uses. Depending on the context we use SHA2 and SHA3 (Keccak). Where it is necessary for safety the digests are truncated to prevent length-extension-attacks.

6.1.4 Key derivation

To improve security and segment the confidentiality of encryption contexts, symmetric encryption keys are derived using HKDF. Key derivation is done with some high entropy constants and dynamic info/context value.

6.1.5 Asymmetric cryptography

For asymmetric encryption we use the popular 25519 curves for both signing and encryption. These curves have great security properties and their implementation is more straight forward than the NIST curves, making them favorable usually. Each account will have two 25519 curve pairs for encryption and signing. The ECDH facilitated by these curves allows two accounts to send each other messages securely. For identity assurance, the keypair used for signing is used to create a signature chain of the account's public keyring. By securely encrypting a fingerprint of a peer account's public singing key, their identity can be checked to be consistent. Utilizing this can prevent the servers from performing man-in-the-middle attacks.

6.2 Account cryptography

During account creation, the client will construct a keyring that will facilitate the cryptography. Simply put, the account's keyring will be encrypted in various ways that ultimately can only be decrypted by knowing the password of the account.

6.2.1 Registration overview

The procedure is based on https://eprint.iacr.org/2015/387.pdf

1. The first step prepares some cryptographical data for initial authentication. The user will enter their email address and password. The email will be sent to the server as registration data.
    - The client shall generate the master-key of 32 bytes with a CSPRNG.
    - The client shall generate a client-salt-input value of 16 bytes with a CSPRNG. This will be sent as registration data.
    - The salt shall be processed. This will facilitate hiding registered emails from snooping and also helps to prevent timing attacks. The salt is a secure cryptographic hash made of the client-salt-input padded to a constant length. This step is executed by the client for registration only. Later, the server will pad the client-salt-input with extra values deterministically to hide unregistered email addresses.
    - The processed salt and the password is derived into two keys using a KDF designed for use with passwords.
        - One of the keys is the authentication-key. Here this will be hashed and sent to the server as registration data.
        - The other key is the encryption-key. This will be used to encrypt the master-key before sending it as registration data.

2. The server will send out a confirmation email in which an URL with a register-token must be followed to continue. By this process the user proves access to the email address and their knowledge of the register process independently of each other. The correctness of the cryptographic primitives created in the first step is also checked.
    - The user will enter the email address once again, which will be sent to the server along with the register-token.
    - The server will respond with a processed-salt when the email is correct for the register-token.
    - The user will enter the password once again. Using the processed salt, the password derivation will take place.
    - The client shall submit the authentication-key, email and register-token to the server. The server will respond with the encrypted-master-key upon correct inputs.
    - The client will use the encryption-key to decrypt the master-key.
    - The client shall generate their asymmetric keys for the keyring. These are an Ed25519 and a Curve25519. The secret parts of these key-pairs shall be encrypted using the master-key.
    - The Ed25519 public key will be the trust-root of the account, so we sign the Curve25519 public key with it.
    - The client will send the authentication-key, email, register-token, encrypted secret keys and public keys of the asymmetric key-pairs and the signature of the Curve25519 to the server. The server will respond with a cryptographic challenge to ensure all constructs and the client operates in good faith and correctly.
    - The client will process the challenge:
        - Using the public Curve25519 key from the challenge, create a shared-secret using their own secret Curve25519 key. This will prove to the server that the client can decrypt and use their Curve25519 key-pair properly.
        - Decrypt the challenge payload using a HKDF derived key from the shared secret. This will prove to the server that the client is capable of key derivation and using a symmetric cipher.
        - Finally it will sign the decrypted challenge payload with their secret Ed25519 key. This will prove to the server that the client can decrypt and use their Ed25519 key-pair properly.
    - The challenge response must arrive in 90 seconds to the server for it to be accepted, where its signature will be validated.

6.2.2 Login overview

1. The user will enter their email address and password.
    - The email address will be sent to the server.
    - The server will respond with a processed-salt. It will return fake salts for incorrect inputs, preventing the detection of actually registered email addresses.
    - Using the processed salt, the password derivation will take place.
    - The client shall generate two session-encryption-keys of 16 bytes each with a CSPRNG.
    - The client shall submit the authentication-key, email and one of the session-encryption-keys to the server. The server will respond with the encrypted-master-key, encrypted-ed2559-secret-key, encrypted-curve25519-secret-key, an encrypted-session-key and a server-public-key upon correct inputs. When the request fails the client is unable to tell if the email, the password or both were incorrect for the login attempt.
    - The client will use the encryption-key to decrypt the master-key.

- The client will use the master-key to decrypt the ed2559-secret-key and curve2559-secret-key.
- The client is now able to create their account's keyring from the decrypted keys.
- Using the account's curve25519 secret key and the server-public-key, the encrypted-session-key will be decrypted by using a HKDF on the shared-secret.
- The session-key shall be set as a cookie for authentication of the client session. NOTE: an additional "httpOnly" cookie is also set by the server response for security.
- The secret keys of the keyring shall be encrypted using a HKDF key from the session-encryption-keys. The encrypted keys and the session-encryption-key that was not sent to the server will be stored in the local storage. The secret keys of the account will only ever be stored encrypted by the session's unique keys generated by the client. By using remote session destruction an offline client may be prevented to recover the encrypted account keys, because the half of the session-encryption-keys that is stored by the server will become unavailable.

6.2.3 Resume overview

1. The user opens up a new browser tab or they logged in with "remember me" and start the browser.
   - The client will look for the cookies and keys to be present. If they are missing then no session-resume may happen.
   - The client shall ask the server for the missing session-encryption-key. The server will validate the client session and return the key if it is still alive.
   - Using the two session-encryption-keys the same HKDF can take place as at the end of login, which will allow the client to decrypt the master-key, ed2559-secret-key and curve2559-secret-key from local storage and re-create their keyring.

6.2.4 Account keyring summary

- The master-key of the account is encrypted by the encryption-key derived from the user's password. The master-key is used to secure all data that is exclusively used by the account: other account-private keys, workspace data, configs, administrative requests to a Nano, etc.
- The private half of the Curve25519 keypair is encrypted by the master-key. This facilitates secure communication between two accounts using ECDH shared-secret generation for chat messages, sharing secret keys with each other, member requests to a Nano, etc.
- The private half of the Ed25519 keypair is encrypted by the master-key. This facilitates identity checking and authenticity guarantees by digital signatures.

6.2.5 Account recovery

An account's password is the the base of the initial key for decrypting an account's keyring. Loosing the password may render the account permanently inaccessible. The keyring cryptography is designed to provide confidentiality for the user, even against the service providing server. Unlike for traditional systems, the support can't help with restoring accounts that can't recall their password.

Specifically, because loosing the password carries such grave danger, an account may reset their password without providing their current one from any active session they have. The password change is always protected by two factor authentication. A malicious person can't lock the owner out of their account using a logged in session, without also gaining access to the owner's emails for example. The Nano client also has the password reset feature to help an owner regain control over their account.

In the case when an account has no active sessions and the password is lost, the only way to regain access is by using a previously saved recovery-key. The recovery-key of the account is actually their unencrypted master-key. It is highly recommended for the user to save it, but the security risk in case of inappropriate storage must also be emphasized. Using the recovery-key, it is possible to gain access to all data stored for the account or transferred by their Nanos if the server colludes with the attacker. For many users, printing the key and not storing it digitally could be ideal. A malicious person that gains access to a recovery-key can't hijack the owner's account solely by that. They'd also need to know the account's email address and gain access to the owner's emails to pass two factor authentication.

In any case, the user is responsible for maintaining secure access to their account by either not forgetting their password, or keeping their recovery-key safe.

6.2.6 Account and session security notes

- all keys and the password KDF input-salt are exclusively generated by the client, without input from the servers
- the password and the encryption-key derived from it never leave the user's machine and they are never stored by the client
- no account-key is placed in permanent storage without encryption
- the session-encryption-keys facilitate remote session destruction while keeping the account keys secure
- the session authentication of the client is protected against XSS stealing, because of a second "httpOnly" cookie
- the webapp is protected from XSS in general using a strict CSP
- all encryption use authenticated encryption, meaning they are all tamper resistant
- all encryption use a HKDF derived key of 64 bytes in AES-SIV (see AES-SIV for details)

6.3 Room cryptography

A room-key is generated at creation by the owner that is encrypted for only them. This key will be shared on a peer-to-peer basis with each account that is invited to the room.

6.3.1 Permissions

The membership and permissions in a room are critically important for security, since these policies are what govern who can access what on the computers by the attached Drives.

The permissions are stored in a cryptographically linked directed acyclic graph, commonly referred to as a blockchain. In addition to them being linked by secure hashing, each block is digitally signed by the owner. This means nobody but the owner can ever create blocks that are valid for the permission graph. This gives the owner of the room and Nano complete control over the room and can ensure the access policy for their Drive.

The Nanos strictly validate the graph when a room is assigned to them. They also save the heads of the graph and will require them to be present or be superseded by new valid blocks. This security measure prevents the server from maliciously omitting the top N blocks and revert some changes to the graph.

We emphasize the difference of this technology and the popular consensus protocols that are commonly referred to as "blockchains". In our case there is no consensus, the permission policy data is always under the sole proprietorship of their owner account. We use the blockchain linkage of data blocks and their digital signature to guarantee tamper resistant storage.

6.3.2 Administrator role

The administrator role in a room effectively means that, the authorized member may send block-chain modification requests to the Nano that is attached to the room, of which the Nano will execute using the owner's account.

This means the administrator role is not functional without a Nano attached to a room. This is necessary, as without a secure client of the owner, nobody is actually capable of modifying the permission blockchain because of the cryptographical challenges.

The owner cannot be removed from a room, ever. They also have all permissions regardless of anything stated in the blockchain.

6.3.3 Anonymous access

Sharing a room using an access link will basically include the unencrypted room-key in the URL, so anyone with it can decrypt the chat and send requests to an attached Nano.

Once an anonymous share URL is public, the room's chat is no longer private. Even if the anonymous access is turned off, "anyone" may have the key indefinitely. At this point the room's chat is protected by the policy enforcement of the server, but not cryptography necessarily. Keep in mind, however that the policy enforcement and client controlled cryptography of the Nano provides complete cryptographic security even at this time for the Drive.

6.4 Nano cryptography

When logging in using a Nano client, the same login procedure is followed that a user login does. Once logged in, the client connects to a server endpoint that routes remote requests to be executed. Depending on the configurations and the incoming requests, the Nano queries the necessary room keys, blockchain and the keyring of peer users for precessing.

6.4.1 Remote request cryptography

All remote requests that arrive to a Nano are always encrypted and go through the following validation process:

- Request metadata provided by the server is validated. This includes the id of the requesting account, the id of the room targeted by the request if any, and various caching and data transfer control information.
- If a room id is provided, but is not served by the Nano, the processing will exit prematurely. This spares the work of decrypting the payload. There is no useful information an adversary may gain from polling the served rooms by this. The server already knows which rooms the Nano serves. Users other than the owner have no ability to send requests to the Nano directly. Since they will request message routing from the server by the room-id, there is no way for them to influence this metadata field, as it is validated and filled in by the server. If they provide a manipulated payload with correctly encrypted but maliciously different embedded room-id, the inconsistency will stop the processing after decrypting the payload and will not reveal served room ids either.
- The decrypting cipher is selected and set up for the account-id provided by the server. There are three distinct encryption configs depending on the requesting account's identity:
  - The owner always encrypts their requests to the Nano with an exclusive encryption using a key derived from their master-key.
  - Anonymous requests are always encrypted using a key derived from the room's pinned key and an ephemeral 25519 keypair. The public key of the anonymous account's keypair is submitted with the request.

  - For any other registered account will need to derive a key from the room's pinned key and their 25519 keypair.

- During the decryption setup, the requesting identity is checked against relevant policies:
  - Anonymous requests will be denied here if the Nano's local policy forbids their execution. (deny_anonymous)
  - Registered accounts (except the owner) will need to pass the identity assurance check of the Nano. This means their identity must be explicitly trusted by the owner or their identity must match the one that was saved prior, to the local Trust-On-First-Use database of the Nano.
- When the decryption of the payload succeeds the account_id provided by the server is guaranteed to be authentic from this point onwards, because the associated keys successfully decrypted the payload from an authenticated ciphertext.
- The server supplied room-id (if any) is checked to be consistent with the one specified in the decrypted payload.
- Local configs and the group config blockchain will be evaluated for the requesting identity selected by the room-id. If the requesting account has no access to the room, or it is denied by a local policy e.g., deny_anonymous, the request processing will exit.
- The request handler key is checked if it is a "mutation". Mutating handlers require the request to supply a session token that prevents malicious or erroneous double submission of requests from executing.

At the point a request is deemed eligible for execution it is guaranteed that:

- The metadata provided by the server and the decrypted request payload is consistent and authentic.
- The server cannot send a request to a Nano other than the designed, because either the decryption would fail or the inconsistency would be detected.
- The request is guaranteed to be made with the knowledge of the required keys.
- The requesting account does not only have the necessary keys, but they have sufficient access granted in the blockchain and the local policy.
- The request is protected against replay-attack if necessary.

6.4.2 Response cryptography and data caching

Responses contain a cryptographic hash that provides a proof to the requesting client that the response received through the servers was made specifically for their request. This prevents malicious servers from performing replay attacks using the Nano responses.

While all requests are encrypted on a peer-to-peer basis between accounts, the responses from a Nano can be encrypted differently. Responses that would benefit from being cached are encrypted by the Nano using deterministically derived keys. The secret values generated by the Nano that are used for deriving the content keys never leave the machine. The servers cache the deterministically encrypted response chunks, allowing multiple users to access the same content without encumbering the network uplink of the Nano.

The server cannot give the cached content to users without explicit authorization of the Nano. When a client requests something from the Nano, the payload is encrypted and the server cannot know if the response would be cached or if it already has the response cached. The Nano must be asked to resolve the request. If the request is authorized the Nano tells the server the cache-key by which the response chunks can be found. Even if the server would guess the response chunks or give all the cached data to any user for a request, they could not decrypt it. The content encryption key is only known by the Nano and it is only shared to clients using peer-to-peer encryption that make authorized requests to that content specifically.

The cached contents are in authenticated ciphertexts, which on by themselves provide data integrity. However these chunks also include checksums of the content that are digitally signed by the Nano. This prevents a malicious server

colluding with a malicious, but authorized account to create tampered contents. Basically all cached content is encrypted with symmetric cryptography, meaning if the server would get hold of the unique key of a content under a specific cache-key, then it could manipulate it. This could happen if a malicious person got access to the room and the content, where a Nano would see them as authorized and share the key for the content with them. Using that key the colluding account could make the server able to tamper with the data chunks. The additional digital fingerprinting and signature of the owner made by the Nano makes this impossible, providing assurance that the cached data was not tampered with.

To break this down:

- Each response carries a proof that it was specifically made for the client's request, the server can't replay previous Nano responses to clients.
- Nano either responds to requests in the same peer-to-peer encryption that the request was made, or using an encryption it has complete control over. (latter is typically used for cache-able responses)
- The source key material used for deriving deterministic keys for cache-able content never leaves the machine the Nano is running on.
- Each cache-able content is derived unique keys by the Nano.
- For a client request to be fulfilled from server cache, the client needs the decryption key of the cached content chunks and the server needs the cache-key of the corresponding responses. Both can only be provided by the Nano. Cached content can't be served by the servers without the Nano explicitly authorizing the request.
- The cached data transfer is guaranteed to be authentic even in the face of maliciously colluding authorized member accounts of the room and the server.