Requirements:

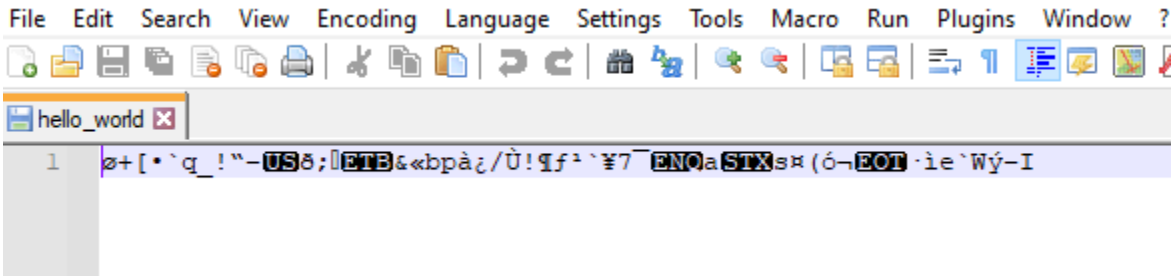
Rust and Cargo both need to be installed on your operating system. See here for more information: https://www.rust-lang.org/tools/install.

Manual Test Plan:

Run “cargo build –bin notes” in the “notes” directory. This will create a binary file “notes” in “notes/target/debug”. Navigate to this directory.

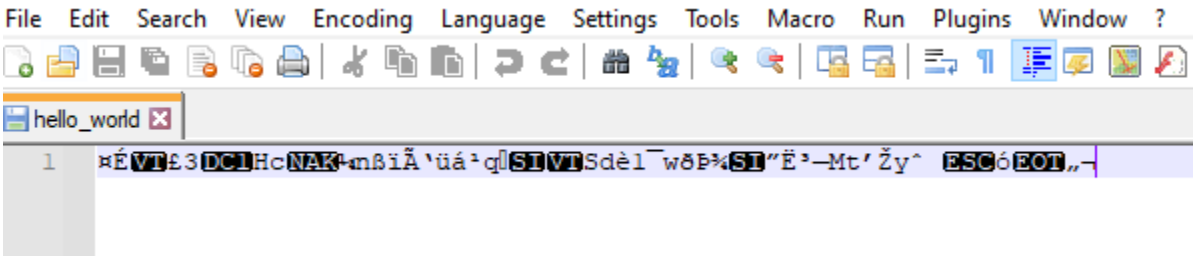
Test 1: File is saved and isn’t readable.

Run the save command using any input. An example is given here: `notes -s -f hello_world -p abcd -I “hello_world!”`. This will create a file called hello_world, which will be encrypted. Open the file and check the contents to ensure it is not readable.



Test 2: Encrypting the same message twice gives different results.

Run the save command again, using the same input as test 1. Open the file and check to see if the contents are different.



Test 3: Human readable output.

Run the load command, using the correct password. An example is given here: `notes -l -f hello_world -p abcd`. This will open the file, decrypt it, and output it in JSON format to the terminal. Make sure that the message is readable.

```
{"string":"hello world!"}
```

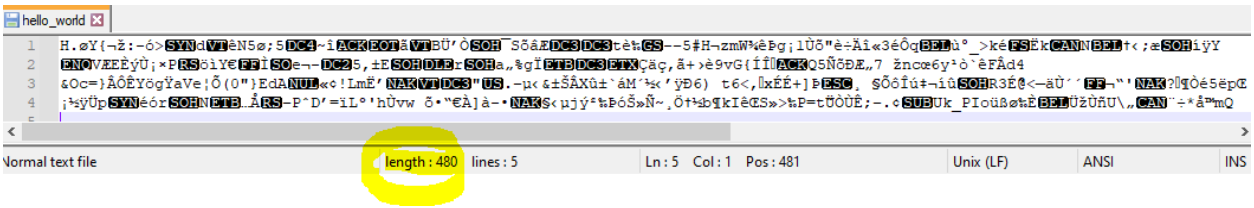
Test 4: File length is padded.

Run the save command with various input lengths. 11 characters is added for JSON formatting. AES256 will always pad the JSON to the nearest 16 bytes and the file will append 16 extra bytes for the IV. Check to see that this padding is maintained using various message lengths.

An example:

`notes -s -f hello_world -p abcd -i "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."`.

Message length is 447 characters. Length after JSON formatting will be 458 characters. AES padding will pad up to the nearest 16 byte interval, which is at 464. An extra 16 bytes will be added for IV, totaling 480.



Test 5: Adding data to tracker appends data.

An example:

notes -s -f tracking_note -p abcd -r weight -d 150

notes -s -f tracking_note –p abcd -r weight -d 151

Run commands like these and ensure that the data field of this weight note appended the new data.

To check the data:

notes -l -f tracking_note -p abcd

Test 6: Adding the same tag twice will not duplicate tag.

An example:

notes -s -f tag_note -p abcd –tag “tag1”

To check if tag was added:

notes -l -f tracking_note -p abcd

Then, run `notes -s -f tag_note -p abcd –tag “tag1”` again.

Check if the tag was not duplicated by running `notes -l -f tracking_note -p abcd`.

Test 7: Executing task commands require complete argument.

Run `notes -s -f task_note -p abcd -t task1`, and verify that the file is not updated and that proper output is shown. That is, the command line argument `-c` must be provided for it to execute.

Test 8: Adding subtask with the same name to different tasks

An example:

`notes -s -f task_note -p abcd -t “task1” -c n` (will add the task, and make it incomplete)

`notes -s -f task_note -p abcd -t “task1” –subtask “subtask1” -c n` (will add the subtask “subtask1” to “task1” and mark it incomplete.)

`notes -s -f task_note -p abcd -t “task2” –subtask “subtask1” -c n` (will add the subtask “subtask1” to “task2” and mark it incomplete.)

To check:

`notes -l -f task_note -p abcd`

Verify that there exist two parent tasks, both with subtask1 as their subtasks, marked incomplete.

Test 9: Disappearing note is still accessible before expiry

Create a note and set an expiry time for 10 seconds. Load it before the time expires and ensure that the note can still load and displays all the information that was put into it.

Expected output of the load will be a JSON string, containing the time of creation and time of expiry.

Test 10: Disappearing notes aren’t accessible after expiry

Create a note and set an expiry timer for 5 seconds. Wait 5 seconds, and then try loading the note. The command line should say that the file has expired, and the file should be gone from the system. Ensure that the file is not in the files folder.

After the expiry, the expected output will be “File has expired!”

Test 11: Filtering based on tag

Create a note and tag it with something. Then, create another note and tag it with something else. Use the –filter-tag command line option with the first tag you created and ensure that the first note is the only one that is output. Likewise, use the second tag and ensure that the second note is the only one that is output.

Running the filter with the first tag will output “File with name {your filename here} contains tag”, followed with a line containing the first note as a JSON string with the first tag and time of creation. Using the second tag will output “File with name {your filename here} contains tag”, followed with a line containing second note as a JSON string with the second tag and time of creation.

Test 12: Filtering based on task

Create a new note and add a task to it with any name. Create a second note with the same password, but with a different task name. Use the –filter-task command using the first task name and check to see

if only the first note is displayed. Likewise, use the command using the second task name and check to see if only the second note is displayed.

Filtering based on the first task will create an expected output of “File with name {your filename here} contains (sub)task”, followed by a line containing the first note as a JSON string with the first task and the time of creation. Filtering based on the second task will create an expected output of “File with name {your filename here} contains (sub)task”, followed by a line containing the second note as a JSON string with the first task and the time of creation.

Test 13: Filtering based on time range

Add a couple of notes, several seconds apart, using any input but the same password, and write down the time when they were written.

Use the –filter-time-start and –filter-time-end command line options and ensure that the notes are excluded if the end time is before the time when the first note was written, and if they are included when it is after the last note was written.

Filtering on time range will have an expected output of “File with name {your filename} is within time range.”, followed by a line containing the note as a JSON string and the input provided. The number of files shown will depend on your time ranges used.

Test 14: Tracker stats

Create multiple notes with trackers. They can have any amount of data added to them.

Then, run `notes -p {your password} –tracker-stat`. The expected output is

“File with name {your filename} has tracker with name {your tracker name}.

Data

[{data1}, {data2},…]

Times

[{time1}, {time2},…]”

An example:

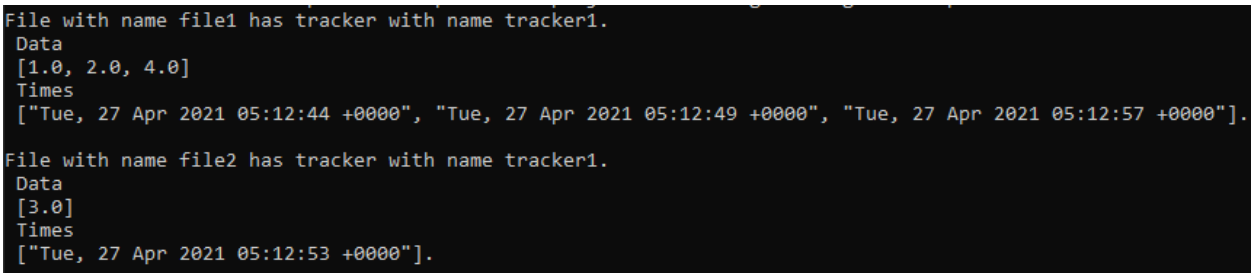
`notes -s -p abcd -f file1 -r tracker1 -d 1`

`notes -s -p abcd -f file1 -r tracker1 -d 2`

`notes -s -p abcd -f file2 -r tracker1 -d 3`

`notes -s -p abcd -f file1 -r tracker1 -d 4`

Will yield:



Ensure that all trackers are output with the correct data and times.

Test 15: Task stats

Create multiple notes with tasks, and some with subtasks. Wait a few minutes and complete them.

Then, run `notes -p {your password} –task-stat`. The expected output is

“File with name {your filename} has task with name {your tracker name}. This task took {time} minutes to complete.”

Additionally, if the task had completed subtasks:

“File with name {your filename} has subtask with name {your tracker name}. This subtask took {time} minutes to complete.”

Ensure that the time taken is correct.

Test 16: Tracker stats with time range

Create multiple notes with trackers. They can have any amount of data added to them. Add tracker over a time period.

Then, run `notes -p {your password} –tracker-stat-start {start time} –tracker-stat-end {end time}`. The expected output is:

“File with name {your filename} has tracker with data

{data}

And times

{times}”

Ensure all the data that was added is in the output, and is properly restricted by the time range.

Test 17: Advanced tracker stats

Create multiple notes with trackers. They can have any amount of data added to them.

Then, run `notes -p {your password} –advanced-stat`. The expected output is

“File with name {your filename} has tracker with name {your tracker name}.

Data

[{data1}, {data2},…]

Times

[{time1}, {time2},…]

File with name {your filename}

Tracker with name {your tracker name}

Average: {average}

Net gain/loss: {net gain/loss}

Median: {median}”

An example:

`notes -s -p abcd -f file1 -r tracker1 -d 1`

`notes -s -p abcd -f file1 -r tracker1 -d 2`

`notes -s -p abcd -f file2 -r tracker1 -d 3`

`notes -s -p abcd -f file1 -r tracker1 -d 4`

Will yield:

```
File with name file1 has tracker with name tracker1.
 Data
 [1.0, 2.0, 4.0]
 Times
 ["Tue, 27 Apr 2021 05:12:44 +0000", "Tue, 27 Apr 2021 05:12:49 +0000", "Tue, 27 Apr 2021 05:12:57 +0000"].

File with name file2 has tracker with name tracker1.
 Data
 [3.0]
 Times
 ["Tue, 27 Apr 2021 05:12:53 +0000"].

File with name file1
Tracker with name tracker1
Average: 2.3333333333333335
Net gain/loss: 3
Median: 2
File with name file2
Tracker with name tracker1
Average: 3
Net gain/loss: 0
Median: 3
```

Ensure the statistics are calculated correctly and all the data is present.