Systems Programming Project 2

April 1, 2025

1 Introduction

CAREFULLY READ THE *ENTIRETY* OF THIS DOCUMENT. DEVIATIONS FROM THE INSTRUCTIONS IN THIS DOCUMENT *WILL* BE PENALIZED.

IF YOU HAVE QUESTIONS OR NEED CLARIFICATION ON SOMETHING IN THIS PROJECT DESCRIPTION, YOU *MUST* INCLUDE THE RELEVANT TEXT FROM THIS PROJECT DESCRIPTION YOU ARE ASKING ABOUT IN YOUR EMAIL, OR WE WILL ASSUME THAT YOU HAVEN'T FULLY READ THE PROJECT DESCRIPTION AND *WILL NOT* RESPOND. LACK OF RESPONSE TO EMAILS THAT DO NOT FOLLOW THIS REQUIREMENT WILL NOT BE CONSIDERED AN EXCUSE FOR DEVIATING FROM THE DIRECTIONS OUTLINED IN THIS DOCUMENT.

2 Hashassin

Now that we have some basics of hashing, etc. down, we are going to make a full blown rainbow table implementation.

A good place to start is by copying your Project 1 workspace.

This is a group project with a maximum of *two* students per group.

NB: Group size is non-negotiable. Attempts to negotiate group size will be considered attempting to acquire an unearned grade.

3 Deliverables

Your submission should be made to GitHub classroom.

- GitHub Classroom link: `https://classroom.github.com/a/WyYhPswz`
- May 1, 2025. **NB: This is a hard cut off and you will not be able to commit to this repository when the due date passes without a late penalty. You should add at least your `HONESTY.md` file to this repository as soon as you accept it to ensure you can do a pull request if you happen to be late.**

4 Instructions

For this project, you will be extending your solution from Project 1 to include a full blown implementation of a rainbow table and corresponding password cracker.

As with Project 1, your binary crate should be in a directory called `cli` and it should compile to a binary named `hashassin`.

Your library crate should be in a directory called `core` and it should be available called `hashassin-core` and usable by other crates via `use hashassin_core::Whatever`.

Your CLI will will support three additional commands: 1) `gen-rainbow-table`, 2) `dump-rainbow-table`, and 3) `crack`.

`gen-rainbow-table` will generate a rainbow table with chains starting from a list of preexisting passwords. `gen-rainbow-table` *must* have several options that can be set:

1. `--num-links`, which specifies the number of links generated chains should have. `--num-links` must be greater than zero and should have a default value of 5.
2. `--threads`, the number of threads to use when generating the rainbow table (format specified below). This value *must* be greater than zero and the maximum value should be the maximum length of an array on whatever system the program is being run on. `--threads` must have a default value of 1.
3. `--out-file`, which is where the generated rainbow table should be saved.
4. `--algorithm`, the hashing algorithm that should be used for this rainbow table. `--algorithm` should have a default value of md5 (i.e., the default hashing algorithm should be md5).
5. `--in-file`, which is a path that has the seed set of passwords that chains should be built from. This file should be in the same format as what's output from Project 1 `gen-passwords`. Each chain in the output rainbow table should begin with one of the passwords specified in `--in-file`. All passwords in the rainbow table should be of the same length as the *first* password in the input file. If the input file contains passwords of varying length, your program should *error* out (*not panic*).

By default, the valid character set for passwords is the same as Project 1: valid ASCII, both caps and lower case, and including punctuation and spaces but *not* including non-printable characters like newline or tabs or bells.

Output file format: Your output file *must* conform to the following format. Non-conformant output will prevent you from earning points.

The file format is consists of a header with a few fields and a data section. The data section begins immediately following the end of the header.

NB: All numbers in the header (e.g., version, lengths, key size, offset) must be *big endian encoded*!

Header format:

- `MAGIC WORD`: The first n bytes of the header should be a utf8 encoded string "rainbowtable" (all lower case).
- `VERSION`: The next byte after the end of the magic word *must* be a version number. Unless otherwise specified in a future update, this should always be `1`.
- `ALGORITHM LENGTH`: The next byte of your output file should be the length of the ASCII encoded string representation of the hashing algorithm used to generate the rainbow table (see next field).
- `ALGORITHM`: The next byte of your output file should include an ASCII string representation of the hashing algorithm that was used to generate the file, in all lowercase. **NB:** this is *not* a null terminated string!
- `PASSWORD LENGTH`: The byte following the name of the hashing algorithm must be the number of characters in the passwords rainbow table was generated from.
- `CHARACTER SET SIZE`: The next 16 bytes of the file should be the character set size of passwords in the rainbow table (i.e., the radix we use in our reduction function).
- `NUMBER OF LINKS`: The next 16 bytes should be the number of links in each chain of the rainbow file.
- `ASCII OFFSET`: The next byte should be any offset (from 0) that passwords in the rainbow table conform to.

The remaining bytes in the file consist of the start and endpoints of each chain in the rainbow table. They should be ASCII encoded bytes, with the first $password\_length/2$ bytes being the start of a given chain and the next $password\_length/2$ bytes being the end of a given chain. There should be no padding between chains.

NB: You can add additional functionality that can restrict or expand the set of characters to be used in generated passwords, but by default, generated passwords should adhere to the above guidelines. You can also do anything else you want with the file format, *except* that your program should always produce output conforming to the above spec by default.

`dump-rainbow-table` will print a human readable version of a rainbow table to the console and *must* have one option that can be set:

1. `--in-file`, which is the path to the rainbow table to dump. This file is *required* and *must* be in `gen-rainbow-table`'s output format.

Output format: `dump-rainbow-table`'s output *must* conform to the following format:

Line 1: "Hashassin Rainbow Table".

Line 1: "VERSION: $VERSION_NUMBER", where $VERSION_NUMBER is the version number in the supplied input file.

Line 2: "ALGORITHM: $ALGORITHM", where $ALGORITHM is the name of the algorithm as specified in the input file.

Line 3: "PASSWORD LENGTH: $PASSWORD_LENGTH", where $PASSWORD_LENGTH is the password length specified in the input file.

Line 4: "CHAR SET SIZE: $CHAR\_SET\_SIZE$" where $CHAR\_SET\_SIZE$ is the number of distinct characters in the valid character set for passwords. E.g., if the only valid character in a password is "a", then $CHAR\_SET\_SIZE$ would be 1. If both "a" and "b" were allowable in passwords, then $CHAR\_SET\_SIZE$ would be 2. If *every* ASCII character was allowable in a password, then $CHAR\_SET\_SIZE$ would be 255. This number is also the radix we use for encoding.

Line 5: "NUM LINKS: NUM_LINKS", where NUM_LINKS is the number of links in the chains of the input rainbow table.

Line 6: "ASCII OFFSET: $ASCII\_OFFSET$", where $ASCII\_OFFSET$ is any offset (from 0) that passwords in the rainbow table conform to.

The remaining lines should be be the chains in the rainbow. Each line should be the start point of a chain and its corresponding end point, separated by a tab (`\t`)

For example::

```
Hashassin Rainbow Table
VERSION: 1
ALGORITHM: md5
PASSWORD LENGTH: 20
KEY SIZE: 26
NUM LINKS: 5
ASCII OFFSET: 97
fasdfasdsljsdffsdlsl\tdffssdsljsdlslasdfaf
lsdflkjdgflkjasitlku\tjaslsdfitlkjkudgflkl
```

`crack` will, given hash file in the format specified in project 1, produce any passwords for those hashes that are found in a pre-computed rainbow table. `crack` *must* have several options that can be set:

1. `--in-file`, which specifies the path to read the rainbow table from.

2. `--out-file`, which, *if present*, will write the output of the command to the specified file, with one pair of hash *hex encoded* and corresponding password separated by the tab character, per line,. If not present, results should be written to `stdout`.

3. `--threads`, the number of threads to use to crack passwords. This value *must* be greater than zero and the maximum value should be the maximum length of an array on whatever system the program is being run on. `--threads` must have a default value of 1.

4. `--hashes`, which is a path to a set of hashes to crack. The input path *must* (by default) have hashes in the format specified in project 1.

You are free to add other options and functionality, but whatever you add should not be required to be set by a user. I.e., it should either be optional or have sensible defaults.

By default, the valid character set for passwords is the same as Project 1: valid ASCII, both caps and lower case, and including punctuation and spaces but *not* including non-printable characters like newline or tabs or bells.

Output should be *hex encoded* hash and its corresponding password, separated by a tab character, with one pair per line.

If there were no passwords found for the input hashes, then `crack` should *error* out with an error message saying "No passwords found."

NB: `crack` should *not* panic if there is no password found. It should *error* out.

5 Grading

You will receive points according to the following list. Please note that there are more than 100 points available.

-1,000 points: If you do not update `CREDITS.md` with the names of your group members, as well as an honest break down of the work each group member did, you will receive **NEGATIVE ONE THOUSAND** points. I.e., it would be next to impossible to get anything higher than a zero on this project.

-1,000 points: If you do not include an `HONESTY.md` with the academic honesty statement from the syllabus, you will receive **NEGATIVE ONE THOUSAND** points.

-1,000 points: If you do not include a `README.md` with information about your project, libraries used, instructions on how run things, etc., you will get **NEGATIVE ONE THOUSAND** points. Your `README.md` should also include the items in the grading rubric that you attempted. Only items you note in your `README.md` file will be graded!

-1,000 points: If running `cargo fmt` results in any change to your repository, you will get **NEGATIVE ONE THOUSAND** points. Be sure to run `cargo fmt`!!!!!!!!!

-1,000 points: Program does not compile to correct binary/library name.

25 points: Program compiles.

25 points: All required functionality of `gen-rainbow-table` is implemented (e.g., setting `--threads` actually uses multiple threads, etc.).

- 5 points: `--num-links`.
- 5 points: `--threads`.
- 5 points: `--out-file`. Output file must conform to specification.
- 5 points: `--in-file`.
- 5 points: `--algorithm`. Support the `md5`, `sha256`, and `sha3_512` algorithms

15 points: All required functionality of `dump-rainbow-table` is implemented.

- 5 points: `--in-file`.
- 10 points: Output conforms to specification.

25 points: All required functionality of `crack` is implemented (e.g., setting `--threads` actually uses multiple threads, etc.).

- 5 points: `--in-file`.
- 5 points: `--out-file`.
- 5 points: `--threads`.
- 5 points: `--hashes`.
- 5 points: Output conforms to spec.

5 points: Comprehensive documentation. Points will be determined by looking at the documentation built when running `cargo doc --document-private-items --no-deps`.

2.5 points: No warnings from `cargo check` with no use of directives that would suppress default warnings (e.g., `#[allow(dead_code)]`)

2.5 points: No warnings from `cargo clippy` with no use of directives that would suppress default warnings (e.g., `#[allow(dead_code)]`)

0.1 points: Support additional hashing algorithms besides md5 (0.1 points per algorithm).

5 points: Support passwords that are greater than 10 characters.

5 points: Handle character set sizes that are greater than 255. E.g., allow unicode in passwords.

2.5 points: Proper error handling. Create and use proper error types in library code and appropriate handling in any user code (e.g., CLI).

2.5 points: No unwraps or expects. Add `#![deny(clippy::unwrap_used, clippy::expect_used)]` to the top of your `main.rs` and `lib.rs`. If running `cargo clippy` doesn't result in any errors, then you get the 5 points.

2.5 points: Add proper logging. Using the `tracing` crate to generate logs. Points will be determined on comprehensiveness of logs, as well as the usage of different log levels.

15 points: Performance report. In your repository, create a file `PERFORMANCE.md` that shows the performance characteristics of your program. E.g., how does it scale with respect to length of passwords? What performance differences are there between md5, sha256, and sha3_512 (and any other algorithms you might test)? How does threading affect performance? This report is not expected to be massive, but it should explain your experimental setup, provide some commands to reproduce your benchmarks, and should have some numbers in it. Plots and tables would be pretty cool. The report is not required to conform to any particular formatting (besides markdown), but please try to make it look decent.

Unlimited points: Cool factor. Points are determined subjectively by me, but if I find anything about your project to be particularly unique, difficult, clever, etc., you can get some extra points. Be sure to point out anything we should pay special attention to in your `README.md` file.

In addition to the above, you must have a README.md to summarize what you did and give any and all instructions on how to run things! We're going to go off what's in the README.md to grade, so make sure you tell us everything you did and test all the instructions, etc.

5.1 Allowed Crates

Generally speaking, you are free to use any crates listed on `https://blessed.rs`.

You can also use the `hex` crate (`https://docs.rs/hex/latest/hex/index.html`) as well as the `num` crate (`https://docs.rs/num/latest/num/`).

It is also ok to use the `bon` crate (`https://docs.rs/bon/latest/bon/`) to help you build more ergonomic APIs (especially using the builder pattern).

Any additional crates not on Blessed.rs need to be approved by Jeremy.

Be sure to note what crates you use in your `README.md`!