Project 2 Documentation

Gavin Dennis | CS 3780-002

Project Environment

I decided to use Java with the _ compiler in Visual Studio Code for my project. This is due to my familiarity with the language as well as its cryptography libraries which have great documentation and tutorials on the internet. Hashing was performed with the *java.security.MessageDigest* class using the SHA-256 algorithm. Salting was performed with a predetermined single-byte that I converted to a string (3A) which I stored in *storedsalt.txt*.

Experiments and Results

I conducted my experiments using randomly generated accounts of size 1, 10, and 100. The passwords were between 5-7 characters long and only utilized lowercase letters. I used *System.currentTimeMillis()* to measure the time it took to crack one password from the list using a brute-force algorithm. I also measured the differences when attempting to crack hashed passwords vs hashed and salted which was quite noteworthy.

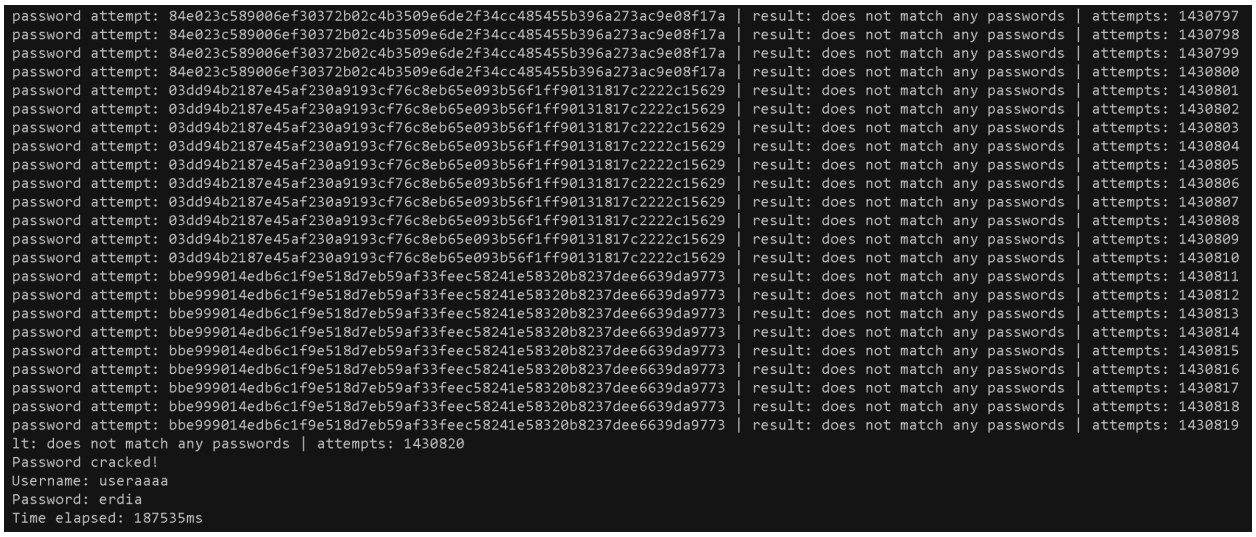
Hashed Password Cracking - 1 Account

When attempting to crack a password from a file of one account, my cracker took over 700,000 attempts and a little over a minute to crack a 5 character password.

```
password attempt: d3b4018f3fd06aea8478c057f476b12008e21a3088c7661b85b3df54e40ca96d | result: does not match any passwords | attempts: 721994
password attempt: cb2d8eded3e62d3ffd941ecc3423ea08af77ab7bd3fe0997ab9ab594c0844727 | result: does not match any passwords | attempts: 721995
password attempt: 058ad46621fed36f7ef6bfa6b80d2c75abf5a0c43a56979db95caebdd7e03d25 | result: does not match any passwords | attempts: 721996
password attempt: 37d71b0440f8057360014985af7a3889d7ac32f774cedede04814ce193690035 | result: does not match any passwords | attempts: 721997
password attempt: 721154a5c43f9c8c494d8d1c1ca3ef90d618dd8ce1a00a8b5631d4a12a50df29 | result: does not match any passwords | attempts: 721998
password attempt: 262844c75d0e0746190d4679f8fd1bdc786b775d3eef71e3bacb65e999a1ff8e | result: does not match any passwords | attempts: 721999
password attempt: bb0f17b9ac35da133a6121a66c7355a0b411f25e16c4f9179fb57359d11899ae | result: does not match any passwords | attempts: 722000
password attempt: 75a19c3902e7be8659bc96639c6378e91f12ffae499827e5c3751e49a33e675e | result: does not match any passwords | attempts: 722001
password attempt: c3363fcc420560bd22d48332f0bb9af2eb525b1d5ff98e6b4507c8006f6b880c | result: does not match any passwords | attempts: 722002
password attempt: eee48f7b6801cff955c1f99467eff2fb9f87c1e0861b60a83cf17dac1a884d64 | result: does not match any passwords | attempts: 722003
password attempt: 229ebe55456f9465d66d0404c1e12298af7c882b004161baf0bd0b88d810ea21 | result: does not match any passwords | attempts: 722004
password attempt: 2e9ebd15f54c66378968bbb56d23243c6770454a559985c066c567a2b44274e3 | result: does not match any passwords | attempts: 722005
password attempt: ad512cbfae57806cca114db2f0f0a0bee404d4792bcb011ada450bc0619ba136 | result: does not match any passwords | attempts: 722006
password attempt: 0cbe9915bc3bd1b7dc8c09e53000fe13bdd5167d79194d2985924dce79b3a7b0 | result: does not match any passwords | attempts: 722007
password attempt: fb0da550469725187f609166a8b4b270e67098c75ae7a250718a4e1b9e513904 | result: does not match any passwords | attempts: 722008
Password cracked!
Username: useraaaa
Password: pbcpb
Time elapsed: 66957ms
```

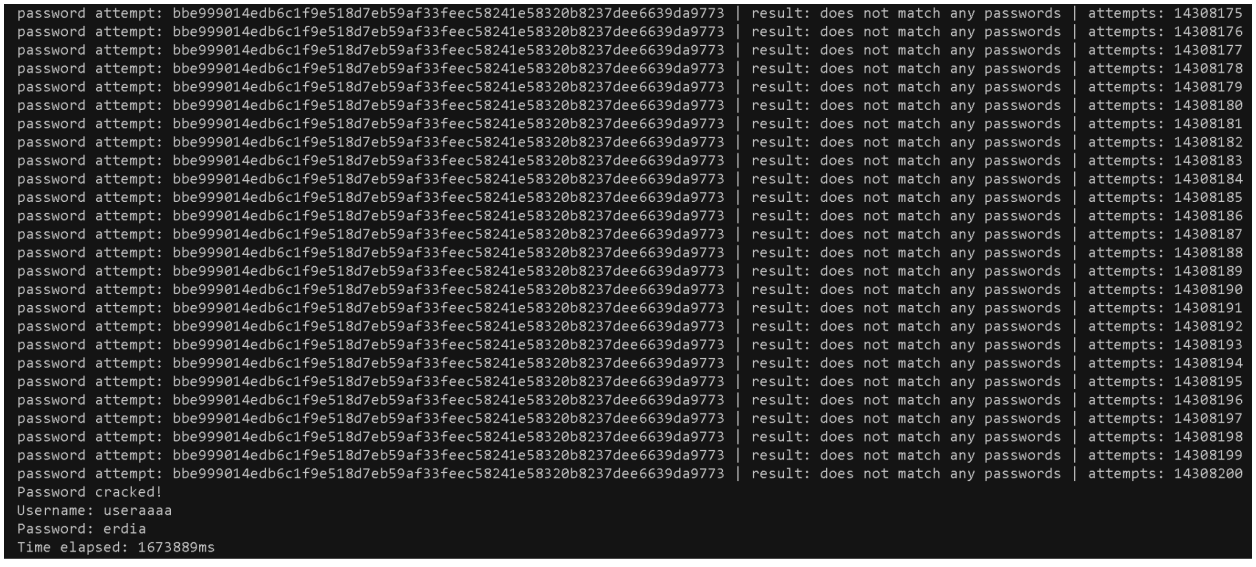
Hashed Password Cracking - 10 Accounts

When attempting to crack one password from a file of ten accounts, my cracker took nearly 1.5 million attempts and over three minutes to crack a 5 character password. This result is consistent with what I would expect from a brute-force attack on a larger dataset.



Hashed Password Cracking - 100 Accounts

When attempting to crack one password from a file of one-hundred accounts, my cracker took well over 14 million attempts and almost half an hour to crack a 5 character password when comparing each result to every password in the file. I realized that the screen output showcases a slight error as the second column should read: result: does not match *current* password.



Hashed + Salted Password Cracking - An Expected Problem

I conducted this experiment using the same plaintext password as the first hash password crack. To generate the cracks, I ran a for-loop with each generated password to add a one-byte

string to the beginning from 00 to FF. This multiplied the amount of attempts by 255, meaning that theoretically it would take around 4 hours and nearly 200 million attempts to crack the single password.

Unfortunately, I did not have the time and computational power to finish running this function, but I can conclude the practicality of adding something as small as two hexadecimal numbers to a password to exponentially improve its security.

```
password attempt: 6A40c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848491
password attempt: 6B40c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848492
password attempt: 6C40c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848493
password attempt: 6D40c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848494
password attempt: 6E40c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848495
password attempt: 6F40c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848496
password attempt: 7040c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848497
password attempt: 7140c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848498
password attempt: 7240c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848499
password attempt: 7340c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848500
password attempt: 7440c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848501
password attempt: 7540c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848502
password attempt: 7640c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848503
password attempt: 7740c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848504
password attempt: 7840c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848505
password attempt: 7940c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848506
password attempt: 7A40c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848507
password attempt: 7B40c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848508
password attempt: 7C40c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848509
password attempt: 7D40c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848510
password attempt: 7E40c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848511
password attempt: 7F40c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848512
password attempt: 8040c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848513
password attempt: 8140c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848514
password attempt: 8240c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848515
password attempt: 8340c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848516
password attempt: 8440c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848517
password attempt: 8540c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848518
password attempt: 8640c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848519
password attempt: 8740c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848520
password attempt: 8840c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848521
password attempt: 8940c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848522
password attempt: 8A40c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848523
password attempt: 8B40c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848524
password attempt: 8C40c9b2afa7258eed5c1bdd408a6ec9df6feb4ba235cbffc2f55ef571a55e836e | result: does not match any passwords | attempts: 24848525
```

Source Code

App.java

```java
// Gavin Dennis
// CS 3780
// Project 2
// Due 2024 November 3

// dependencies
import java.io.File;  // for file handling
import java.io.FileWriter;  // for writing to files
import java.io.IOException;  // for exception handling when writing to
files
import java.util.Random;
import java.util.Scanner;  // for user input and file reading
import java.util.regex.Matcher;  // for username/password pattern checking
import java.util.regex.Pattern;  // for username/password pattern checking
import java.security.MessageDigest;  // for password hashing
import java.util.ArrayList;  // for resizable array for password cracking

// exception handling
import java.io.FileNotFoundException;
import java.security.NoSuchAlgorithmException;
import java.io.UnsupportedEncodingException;

public class App {
    public static void cPrint(String str) {
        // c-style print function for easier screen logging

        System.out.println(str);
    }

    public static void debug(String str) {
        // same as cPrint but conditional based on whether testing or not
        boolean TEST = false;
        if (TEST) { cPrint(str); }
    }

    public static String bytesToHex(byte[] bytes) {
        // convert message digest to hexadecimal string
```

```
        StringBuilder hexString = new StringBuilder(2 * bytes.length);

        for (byte b: bytes) {
            hexString.append(String.format("%02x", b & 0xff));
        }

        return hexString.toString();
    }

    public static void login() throws NoSuchAlgorithmException,
UnsupportedEncodingException {
        // handles login by getting user input and validating with all
three password files

        // declare variables and input scanner
        Scanner input = new Scanner(System.in);  // scanner for user input
        String userInput = "";  // string for inputted username
        String pwInput = "";  // string for inputted password
        boolean isValid = false;

        while (!isValid && !userInput.equals("menu")) {
            // get username attempt from user
            cPrint("Enter username (or type 'menu' to go back to the
menu): ");
            userInput = input.nextLine();

            // break off to the menu per user's request
            if (userInput.equals("menu")) { continue; }

            // get password attempt from user
            cPrint("Enter password: ");
            pwInput = input.nextLine();

            // validate login
            isValid = validateLogin(userInput, pwInput);

            if (isValid) {
                cPrint("Login Success!\n");
            }
```

```
            else {
                cPrint("ERROR: Invalid username or password. Please try
again.\n");
            }
        }
    }

    public static boolean validateLogin(String userInput, String pwInput)
throws NoSuchAlgorithmException {
        // actual logic for validating login


        String userFromFile = "";  // string for validating username from
password files
        String pwFromFile = "";  // string for validating username from
password files
        String hashedPassword = "";  // string for storing user inputted
password after it's hashed

        int lineNumber = 0;  // line number to ensure password is
connected to correct username

        String [] passwordFiles = new String[] {"plaintext.txt",
"hashed.txt", "salt.txt"};
        boolean [] validationArr = new boolean[] {false, false, false};
// array of booleans for validating each password file

        // validate username and password
        for (int i = 0; i < 3; i++) {
            lineNumber = 0;
            userFromFile = "";
            try {

                // open password file and search each line for matching
username
                Scanner file = new Scanner(new File(passwordFiles[i]));

                while (!userFromFile.equals(userInput) &&
file.hasNextLine()) {
                    lineNumber++;
```

```
                    // only check odd number lines for usernames as the
even ones will have passwords
                    if (lineNumber % 2 == 0) { continue; }
                    else { userFromFile = file.nextLine(); }
                }

                // validate the username for the given file if it checks
out
                if (userFromFile.equals(userInput)) {

                    // attempt to validate the password
                    pwFromFile = file.nextLine();
                    // cPrint(pwInput + " and " + pwFromFile);

                    if (i == 0) {  // validate plaintext password
                        if (pwFromFile.equals(pwInput)) { validationArr[i]
= true; }
                        // else { cPrint("Failed at plaintext comparison:
" + pwFromFile + "\n" + pwInput + "\n"); }
                        cPrint("Validation 1: " + validationArr[i]);
                    }
                    else if (i == 1) {  // validate hashed password
                        // hash inputted password
                        MessageDigest sha256 =
MessageDigest.getInstance("sha-256");
                        byte [] digest= sha256.digest(pwInput.getBytes());
                        hashedPassword = bytesToHex(digest);

                        // compare hashed input password to file password
                        if (pwFromFile.equals(hashedPassword)) {
validationArr[i] = true; }
                        // else { cPrint("Failed at hashed password
comparison: " + pwFromFile + "\n" + hashedPassword + "\n"); }
                        cPrint("Validation 2: " + validationArr[i]);
                    }
                    else if (i == 2) {  // validate hashed+salt password

                        // get salt from file
```

```
                        Scanner saltFile = new Scanner(new File("stored
salt.txt"));
                        String salt = saltFile.nextLine();
                        saltFile.close();

                        // compare hashed+salted input password to file
password
                        if (pwFromFile.equals(salt + hashedPassword)) {
validationArr[i] = true; }
                        // else { cPrint("Failed at salted password
comparison: " + pwFromFile + "\n" + salt + hashedPassword + "\n"); }
                        cPrint("Validation 3: " + validationArr[i]);
                    }
                }

                // close the file
                file.close();
            }
            catch (FileNotFoundException e) {
                cPrint("ERROR: Unable to validate password; No password
file.");
                e.printStackTrace();
            }
        }

        // return the result of the three validations
        return (validationArr[0] && validationArr[1] && validationArr[2]);
    }

    public static void createAccount() throws NoSuchAlgorithmException,
UnsupportedEncodingException, FileNotFoundException {
        /* handles account creation by getting user input, validating
        with string length, regex pattern checking, and string comparison,
        then attempts to write to file with writeDataToFiles function */

        Scanner input = new Scanner(System.in);  // scanner for user input

        String unAttempt;  // a string that contains the user's inputted
username attempt;
```

```
        String pwAttempt;  // a string that contains the user's inputted
password attempt;

        String username = "";  // a string that contains the user's
validated ID; 5-10 character length with only alphabetic characters
        String password = "";  // a string that contains the user's
validated password; only lowercase alphabetic characters

        String repeat;  // compared to password to make sure they match

        Pattern unPattern = Pattern.compile("^[a-zA-Z]+$");  // regex
pattern for validating inputted username
        Pattern pwPattern = Pattern.compile("^[a-z]+$");  // regex pattern
for validating inputted password

        int minSize = 5, maxSize = 10;  // integers to validate size of
passwords
        int userSize = 10;  // length of all usernames

        boolean usernameValidation = false;  // boolean to loop input
while input isn't valid
        boolean passwordValidation = false;  // boolean to loop input
while input isn't valid

        // message to user to explain username input
        cPrint(
            ". . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . "
+
            "\nCreate an Account\n"
        );
        cPrint(
            "* * * * * Username * * * * *\n" +
            "Username must be 10 characters in length and only use
alphabetic characters [a-z] and [A-Z]." +
            "Please enter your new username: "
        );

        // get input from user and validate
        while (!usernameValidation) {
```

```
            // get username string
            unAttempt = input.nextLine();

            // validate username input
            // first check the string length
            if (unAttempt.length() == userSize) {
                // if string length checks out, run the regex matcher
                Matcher matcher = unPattern.matcher(unAttempt);
                if (matcher.matches()) {
                    // assign the validated strinmjg to username
                    usernameValidation = true;
                    username = unAttempt;
                }
                else {
                    cPrint("\nInput contains non-alphabetic characters.
\nPlease try again. \n");
                }
            }
            else {
                cPrint(
                    "\nUsername is not specified length (10
characters).\n" +
                    "Please try again. \n"
                );
            }
        }

        // print out username validation message
        cPrint("\nUsername successfully validated.\n");

        // get password from user

        // message to user to explain password input
        cPrint(
            "* * * * * Password * * * * *\n" +
            "Password must be 5-10 characters in length and only use
lowercase alphabetic characters [a-z]." +
            "Please enter your password: "
        );
```

```
        // get input from user and validate
        while (!passwordValidation) {

            // get password string
            pwAttempt = input.nextLine();

            // validate password input
            // make sure the password isn't the same as the username
            if (pwAttempt.equals(username)) {
                cPrint("\nPassword cannot be the same as username. \n" +
"Please try again. \n" + "\nPlease enter your password: ");
            }
            // check the string length
            else if (pwAttempt.length() <= maxSize && pwAttempt.length()
>= minSize) {
                // if string length checks out, run the regex matcher
                Matcher matcher = pwPattern.matcher(pwAttempt);
                if (matcher.matches()) {
                    // prompt user for password confirmation
                    cPrint("\nConfirm password: ");
                    repeat = input.nextLine();

                    // check if password input is the same confirmation
                    if (pwAttempt.equals(repeat)) {
                        passwordValidation = true;
                        password = pwAttempt;
                    }
                    else {
                        cPrint("\nPasswords do not match. \n" + "Please
try again. \n" + "\nPlease enter your password: ");
                    }
                }
                else {
                    cPrint("\nInput contains non-alphabetic characters or
capital letters. \nPlease try again. \n");
                }
            }
            else {
                cPrint(
```

```
                    "\nPassword is not within specified length limit (5-10
characters).\n" +
                    "Please try again. \n"
                );
            }
        }

        cPrint("\nPassword successfully validated. Attempting to write to
files");

        // end the function by closing the scanner and attempting to
append the data to 3 password files
        writeDataToFiles(username, password, true);
    }

    public static void writeDataToFiles(String username, String password,
boolean append) throws NoSuchAlgorithmException,
UnsupportedEncodingException, FileNotFoundException {
        /* takes validated username and password as arguments and writes
them to three files
            1. plaintext username password pair
            2. username and hashed password
            3. username, salt, and hashed (password + salt)
        */

        // get hash and salt
        // hash
        MessageDigest sha256 = MessageDigest.getInstance("sha-256");
        byte [] digest= sha256.digest(password.getBytes());
        String hashedPassword = bytesToHex(digest);

        // get salt from 'stored salt.txt' and convert it from byte form
to string
        Scanner file = new Scanner(new File("stored salt.txt"));
        String salt = "";
        if (file.hasNextLine()) { salt = file.nextLine(); }
        file.close();

        // try writing to file
        try {
```

```
            FileWriter plainText = new FileWriter("plaintext.txt",
append);
            FileWriter hashText = new FileWriter("hashed.txt", append);
            FileWriter saltText = new FileWriter("salt.txt", append);

            // write to files
            plainText.write(username + "\n" + password + "\n");  //
plaintext password
            hashText.write(username + "\n" + hashedPassword + "\n");  //
hashed password
            saltText.write(username + "\n" + salt + hashedPassword +
"\n");  // hashed with salt

            // close the file writers
            plainText.close();
            hashText.close();
            saltText.close();
        }
        catch(IOException e) {
            System.out.println("ERROR: Couldn't print to file.");
            e.printStackTrace();
        }
        System.out.println("Username and Password saved.\n");
    }

    public static void generatePasswords() throws
NoSuchAlgorithmException, UnsupportedEncodingException,
FileNotFoundException {
        /* task 2

        takes in user parameters for password length and users and
generates them
        to a file */

        // declare variables
        int passMin = 5, passMax = 10, accounts = 1;
        String username = "user";  // username prefix (useraaaa -
userzzzz)

        // scanner for user input
```

```
        Scanner input = new Scanner(System.in);

        // message to user
        cPrint("Password generator: \n" + "Enter a password length range
and an amount of accounts greater than or equal to one.\n\n");

        // collect data from user
        cPrint("Enter the minimum amount of characters for passwords (at
least 5, no more than 9): ");
        passMin = input.nextInt();
        passMin = (passMin < 1) ? 1 : passMin;
        passMin = (passMin > 9) ? 9 : passMin;

        cPrint("Enter the maximum amount of characters for passwords (at
least 6, no more than 10): ");
        passMax = input.nextInt();
        passMax = (passMax > 10) ? 10 : passMax;
        passMax = (passMax < 6) ? 6 : passMax;

        // make sure max is greater than min
        if (passMax < passMin) { passMax = passMin+1; }

        cPrint("Enter the number of accounts (at least 1): ");
        accounts = input.nextInt();
        accounts = (accounts < 1) ? 1 : accounts;

        // generate the accounts
        cPrint("\nGenerating accounts...\n");
        char[] suffix = {'a', 'a', 'a', 'a'};  // suffix for username
'userxxxx'
        while (accounts > 0) {
            // increment username by letter and create every password
combination, then write to file
            username = "user";
            String password = "";

            // set the username string
            username = username + suffix[0] + suffix[1] + suffix[2] +
suffix[3];
```

```
            // generate random password for account
            Random random = new Random();
            int passwordLength = 0;
            String pwCharacters = "abcdefghijklmnopqrstuvwxyz";  // a
string of every possible password character
            passwordLength = random.nextInt(passMax - passMin) + passMin;
            for (int i = 0; i < passwordLength; i++) {
                int nextChar = random.nextInt(26);
                password += pwCharacters.charAt(nextChar);
            }

            // overwrite password file with generated data
            writeDataToFiles(username, password, true);

            // increment through every username combination
            if (suffix[3] == 'Z') {
                if (suffix[2] == 'Z') {
                    if (suffix[1] == 'Z') {
                        if (suffix[0] == 'Z') {
                            accounts = 0;
                        }
                        else if (suffix[0] == 'z') { suffix[0] = 'A'; }
                        else { suffix[0]++; }
                    }
                    else if (suffix[1] == 'z') { suffix[1] = 'A'; }
                    else { suffix[1]++; }
                }
                else if (suffix[2] == 'z') { suffix[2] = 'A'; }
                else { suffix[2]++; }
            }
            else if ( suffix[3] == 'z' ) { suffix[3] = 'A'; }
            else { suffix[3]++; }

            // increment accounts left to make
            accounts--;
        }

        // print success message
        cPrint("\nSuccessfully generated usernames and passwords.\n");
    }
```

```
    public static void passwordCracker() throws FileNotFoundException,
NoSuchAlgorithmException, InterruptedException {
        // task 3
        /* attemps to crack a password to a known password file by
        brute-forcing strings and applying hashing algorithm */

        // declare variables and scanner for user input
        Scanner input = new Scanner(System.in);  // scanner for user input
        int openSalt = -1;  // integers that decides which password file
should be opened: 1 for hashed, 2 for salted
        int maxSize = -1;  // max password length inputted from user
        int attempts;  // attempts it takes to crack password;

        ArrayList<String> userArr = new ArrayList<>();  // resizable
string array to store all usernames from password file
        ArrayList<String> pwArr = new ArrayList<>();  // resizable string
array to store all usernames from password file

        char [] crackChars = {'a','a','a','a','a','A','A','A','A','A',
'A'};  /*character array for cracking passwords; 'A' represents

an ignored character since passwords only have lowercase letters */
        int crackIndex = 0;  // index of crackChars that is incremented
after failed crack attempt

        String crack = "";  // concatenation of characters in crack chars
array; plaintext password
        String hashCrack = "", saltCrack = "";  // string for crack with
hashed algorithm and salt
        String salt = "";  // salt to be added to each hashCrack to form
saltCrack

        Boolean cracked = false;  // signifies that password has been
cracked

        long start, end, totalTime;  // long ints used to measure the time
in ms of the cracking process

        // message to user
```

```
        cPrint("Password cracker:\n" + "Enter which file to examine, then
enter username to try and crack password.\n");

        // prompt user for file
        while (openSalt != 0 && openSalt != 1) {
            cPrint("\nEnter which file to examine ('0' for 'hashed
passwords', '1' for 'salted + hashed passwords')");
            openSalt = input.nextInt();
        }

        // prompt user for password size
        while (maxSize < 5 || maxSize > 10) {
            cPrint("\nEnter max password size (Must be greater than or
equal to 5 and less than or equal to 10):");
            maxSize = input.nextInt();
        }

        // open password file and extract inputted usernames and hashed
passwords
        String filename = (openSalt == 1) ? "salt.txt" : "hashed.txt";
        Scanner file = new Scanner(new File(filename));
        while (file.hasNextLine()) {
            userArr.add(file.nextLine());
            if (file.hasNextLine()) {
                pwArr.add(file.nextLine());
            }
        }

        // close the password file
        file.close();

        // attempt to crack passwords with every string combination
        cPrint("\nAttempting to crack passwords...\n");
        start = System.currentTimeMillis();
        attempts = 0;
        while (!cracked) {
            // configure current crack
            crack = "";
            for (int i = 0; i < crackChars.length; i++) {
                if (crackChars[i] != 'A') { crack += crackChars[i]; }
```

```
            }

            // hash the plaintext crack attempt and add salt
            MessageDigest sha256 = MessageDigest.getInstance("sha-256");
            byte [] digest= sha256.digest(crack.getBytes());
            hashCrack = bytesToHex(digest);

            // compare hashed/salted password with every password in file
            for (int i = 0; i < pwArr.size(); i++) {

                // attempt to crack hashed file
                if (openSalt == 0) {
                    // increment attempts
                    attempts++;

                    // compare current hash with password file
                    if (hashCrack.equals(pwArr.get(i))) {
                        cPrint("Password cracked! \nUsername: " +
userArr.get(i) + "\nPassword: " + crack);
                        cracked = true;
                        break;
                    }
                    else { debug("password attempt: " + hashCrack + " |
result: does not match any passwords | attempts: " + attempts); }
                }

                // attempt to crack salt + hashed file
                else if (openSalt == 1) {
                    // configure every one byte possible salt (00-FF)
                    for (int j = 0; j <= 255; j++) {
                        // increment attempts
                        attempts++;

                        // configure salt
                        salt = (Integer.toHexString(j)).toUpperCase();  //
convert j to two digit uppercase hex string
                        if (salt.length() == 1) { salt = "0" + salt; }
                        saltCrack = salt + hashCrack;

                        // compare salted password with password file one
```

```
                    if (saltCrack.equals(pwArr.get(i))) {
                        cPrint("Password cracked! \nUsername: " +
userArr.get(i) + "\nPassword: " + crack);
                        cracked = true;
                    }
                    else { debug("password attempt: " + saltCrack + "
| result: does not match any passwords | attempts: " + attempts); }
                }
            }
        }

        // reconfigure plaintext crack
        // I apologize for this unholy structure. I couldn't find a
better way to configure the brute-force passwords
        if (crackChars[0] < 'z') { crackChars[0]++; }
        else {
            crackChars[0] = 'a';
            if (crackChars[1] < 'z') { crackChars[1]++; }
            else {
                crackChars[1] = 'a';
                if (crackChars[2] < 'z') { crackChars[2]++; }
                else {
                    crackChars[2] = 'a';
                    if (crackChars[3] < 'z') { crackChars[3]++; }
                    else {
                        crackChars[3] = 'a';
                        if (crackChars[4] < 'z') { crackChars[4]++; }
                        else {
                            crackChars[4] = 'a';
                            if (crackChars[5] == 'A') { crackChars[5]
= 'a'; }
                            else if (crackChars[5] < 'z') {
crackChars[5]++; }
                            else {
                                crackChars[5] = 'a';
                                if (crackChars[6] == 'A') {
crackChars[5] = 'a'; }
                                else if (crackChars[6] < 'z') {
crackChars[6]++; }
                                else {
```

```
                                        crackChars[6] = 'a';
                                        if (crackChars[7] == 'A') {
crackChars[5] = 'a'; }
                                        else if (crackChars[7] < 'z') {
crackChars[7]++; }
                                        else {
                                            crackChars[7] = 'a';
                                            if (crackChars[8] == 'A') {
crackChars[5] = 'a'; }
                                            else if (crackChars[8] < 'z')
{ crackChars[8]++; }
                                            else {
                                                crackChars[8] = 'a';
                                                if (crackChars[9] == 'A')
{ crackChars[5] = 'a'; }
                                                else if (crackChars[9] <
'z') { crackChars[9]++; }

                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }

            // get time elapsed
            end = System.currentTimeMillis();
            totalTime = end - start;
            cPrint("Time elapsed: " + totalTime + "ms");
        }

        public static void main(String[] args) throws Exception {

            // create scanner for user input
            Scanner input = new Scanner(System.in);
```

```
        // prompt user for function: login or create account
        int choice = -1;
        while (choice != 9) {
            cPrint(
                "\nChoose an option: \n" +
                "\t1: Login\n" +
                "\t2: Create an Account\n" +
                "\t3: Generate password files\n" +
                "\t4: Password Cracker\n" +
                "\n\t9: Exit program"
            );

            choice = input.nextInt();
            switch (choice) {
                // call whichever function the user picks

                case 1:
                    login();
                    break;
                case 2:
                    createAccount();
                    break;
                case 3:
                    generatePasswords();
                    break;
                case 4:
                    passwordCracker();
                    break;
                case 9:
                    break;
                default:
                    cPrint("Please select '1', '2', or '3' and press
enter.\n");
                    break;
            }

            // reset choice and repeat
            if (choice != 9) { choice = -1; }
        }
```

```
        // terminate the program
        input.close();
        System.exit(0);
    }
}
```