Siyuan Zhang, Nisarg Joshi, Robby Dailey, Samuel Bihinda
CMSC 414: Section 0201
April 30, 2021

HW9 Build-It: Team Anonymous Design Document

System Design:

As a team we first agreed to program our gradebook wrapper using JAVA. We then decided to utilize JSON (JavaScript Object Notation) for our gradebook layout. We choose JSON because it is a lightweight data-interchange format that encodes objects in a single string. JSON makes it easy to read, write, and parse data which is necessary for manipulating a gradebook. Since JSON is a collection of name/value pairs, we used it to represent arrays, strings, and number objects for the gradebook's respective assignments, students, and grades. The following JSON text is an example of our gradebook configuration:

```
{
  "name": "gradebook1",
  "assignments": [
    {
      "name": "HW1",
      "points": 100,
      "weight": .4
    },
    {
      "name": "HW2",
      "points": 100,
      "weight": .6
    }
  ],
  "students": [
    {
      "firstname": "John",
      "lastname": "Doe",
      "grades": [
        {
          "assignment": "HW1",
          "grade": 88
        },
        {
          "assignment": "HW2",
          "grade": 0
        }
      ]
    },
```

```
    {
      "firstname": "Chris",
      "lastname": "Jones",
      "grades": []
    }
  ]
}
```

Our gradebook JSON object has three properties: the “name”, “assignments”, and “students”. The “assignments” property is an array of the assignment objects in the gradebook (with properties of the assignment name, points, and weight). The “students” property is an array of student objects (each has properties for the first name, last name, and grades array which has objects with the assignment and grade). Our gradebook.java code takes care of all the JSON manipulation along with a ToString() method that prints the JSON.

The helper code in Helper.java hosts all the code we use for creating, securing, saving, and retrieving a gradebook file. Since our gradebook.java code manages and provides the actual plain text JSON, helper.java job is to ensure that it is not actually saved in plain text. When it comes to encryption and signing we believed that utilizing RSA was the best route.We relied on JAVA’s java.security package to provide us with powerful and vetted cryptography functions. Our helper.java code will generate a random RSA key pair (public and private). The public key is used to encrypt the plain text gradebook JSON. We then use the private key to sign that encrypted data. We then write both the signature and the encrypted data in a new gradebook file with the specified filename. Even though RSA has unique private and public keys, in our implementation we only print out the private key (which happens to include the data to retrieve the public key) encoded in base64 string for convenience then converted to hexadecimal format. When it comes to reopening the gradebook file, the user must provide that key back so our program can get back the private and public keypair. After retrieving the keys we can verify the signature vs the encrypted data with the private key. After it is verified we can decrypt the data with the private key to get the gradebook JSON in plain text. Our gradebook class also provides

a constructor to recreate the gradebook with the JSON plain text to make the reconstruction simple.

Our gradebookadd program and gradebookdisplay mainly utilizes Java regex to parse the commands. It checks the format of the input with a regex matcher and makes sure the input is valid before calling opening and saving the gradebook. gradebookadd specifically for example after parsing and validating the name and key, the program switches on the five sub-commands {-AA, -DA, -AS, -DS, -AG} to ensure only one command will be run. Lastly, it loops through the rest of the input command, checks to ensure the specific arguments are provided and takes the last value. The program uses the parse_cmdline function to parse the input and return the parameters. If there is anything wrong with the command line, the function will return null and the program will exit with 255 when given a null. Both programs lastly calls the Gradebook.java functions with the corresponding parameters.

Threats:

1) A potential attack is reading the raw contents of the gradebook file using a text editor. We don't want unauthorized users to read the gradebook and see its details. Since the underlying structure is JSON anyone can see the assignments and grades if it is in raw json text. We were able to prevent this attack by encrypting the gradebook JSON text using the well tested RSA encryption package that java.security provides. The encryption function utilizes the public key generated and produces a line of data that is not understandable when the file is reopened in a text editor. Essentially, the JSON is no longer in plain text.

2) An attacker might also attempt to generate the original key using the setup command. This attack is prevented because we utilized the SecureRandom class that java provides as a parameter for our key generation function. No matter what, the key generated will be random every time the setup program is run. The original key associated with the gradebook file cannot be retrieved by running setup again. This prevents the attacker from simply generating the original key and using it to decrypt the gradebook and act as if the attacker was the creator of the gradebook.
3) Another potential attack is replacing the signature of the encrypted gradebook JSON with the attacker's own signature using their key. This attack will fail because right after we verify the signature in our OpenGradeBook function, we try to decrypt the data with the same key that was used to verify the data and the key will fail to decrypt that data with the attack's key and an exception is thrown. Only the original key can verify the encrypted json data and also decrypt it. Replacing the signature will not be enough.
4) A potential attack that we also prevented was a buffer overflow attack that an attacker might try. Our decision to program in Java instead of C was to avoid this attack since java Strings are based on char arrays and Java automatically checks array bounds every time. This stops an attacker from manipulating the stack in any way thus buffer overflows are only possible in unusual scenarios like the JAVA compiler being broken or something.

Member Contributions:

Siyuan Zhang: Designed gradebookadd.java which handles all of the parsing of the command arguments for adding or deleting assignments, students, and grades. Ensures all the necessary parameters are there before passing them to the Gradebook functions to manipulate the actual Gradebook data.

Nisarg Joshi: Completed the gradebookdisplay.java which handles all of the parsing of the command arguments for displaying the gradebook. Ensures all the necessary parameters are there before passing them to the Gradebook functions to display.

Robby Dailey: Designed the Gradebook.java which handles all of the JSON manipulation including adding/deleting assignments, students, and grades. Also handles illogical command parameters when manipulating the gradebook. Included the ability to get the JSON as a single string (ToString()) and also rebuild the Gradebook by passing in the plain text JSON in the constructor. Helped incorporate Gradebook.java functions in the gradebookadd and gradebookdisplay by calling the appropriate functions with the necessary variables.

Samuel Bihinda: Completed the command line program that is executed by running setup in setup.java (3.1) which creates a new gradebook file and prints the generated key. Designed the Helper.java class which handles all of the logic needed to generate new keys, encrypt the gradebook, decrypt the gradebook, save the gradebook, and reopen the gradebook. Took care of all the cryptological security needs for the gradebook to ensure only the key holder(s) can read their created gradebook. These helper functions are used by setup.java, gradebookadd.java and gradebookdisplay.java for retrieving and saving the gradebook.