Software Design Document

Language choice

While C offers unparalleled speed and memory manipulation, our team opted to use Java.

Upsides of Java include but are not limited to:

- Ease of use
  - Memory management handled by the built-in garbage collector, easier syntax, and easy to implement object oriented paradigm make the source code far more concise and easier to collaborate on between teammates.
  - Not to mention, the very extensive standard library
- Security
  - Memory management handled by the built-in garbage collector significantly reduces (but does not eliminate) the risk of temporal and spatial compromise during the program's runtime.
- Third party library support
  - Java being a modern language supports many modern build systems and critical infrastructure, and as a result has a bustling ecosystem of hardened cryptography and argument parsing libraries perfect for our use case.
- Multi-platform feature parity
  - Because of Java's hybrid model of being compiled for the Java Virtual Machine but interpreted by the JVM to run on several platforms, we can scale both the Gradebook server and

Gradebook object

Our Gradebook class utilizes three other classes: Student, Assignment, and Grade. Student represents a student, with a firstname and lastname. Assignment represents an assignment, with a name, total number of points, and weight. And Grade represents a student’s grade, with an associated Assignment object and number of points for that grade.

Our Gradebook object holds a map, with Student objects as the keys, and a list of Grade objects as the value associated with a key, that is to say, we have a list of grades for every student.

In addition, our Gradebook object holds a list of Assignment objects, since we need to keep track of what assignments are in the gradebook.

The Gradebook includes methods to add a new assignment, delete an assignment for all students, add a new student, delete a student, and add a grade for an existing student and assignment. Also, there are methods for printing out grades of all students for a particular assignment, printing out all grades for a particular student, and printing final grades for all students.

Add and Display Programs

Our code contains two programs, gradebookadd and gradebook display. These programs are used to update the gradebook and display the gradebook that was initialized when the setup program is run.

The bulk of these classes deal with the parsing of the command line and verifying that the command is in fact valid. A bulk of the work is done by calls to methods in the Gradebook class, e.g -AA is handled by addAssignment, -PA is handled by printAssignment, etc. The regex for names is also handled by the Gradebook class, the parsing of the command line is only concerned with the order and presence of the arguments, e.g -FN, -N, -K, etc.

The parsing is done through a simple for-loop which initially checks for a command (-N, -K, -AA, etc.) and then updates the corresponding variable to the next argument. If an argument is out of order or not present then the code will error and exit.

Gradebook security

The two vulnerable points of the gradebook are the gradebook file itself being intercepted by malicious parties as well as the secret key used. We operate under the assumption that parties won't have access to the key, so the majority of our security needs to be around the gradebook object.

Our current security measures are as follows: A valid gradebook object is created and or accessed via an encryptObject method that accepts a key and Java serializable object. We run this method both when a new gradebook is created in the setup program as well as after any read/write operations are done through the add or display programs. Decryption occurs at the start of an add or display operation with the encryption occurring shortly after completion.

Setup and authenticated encryption

In the setup program each gradebook gets a string of bytes for the key. This key is a 256 bit key meant to be used with AES. This length is larger than the tag length we use for integrity later on as a tradeoff for speed. Modification and validation of each gradebook block occur more often than a one time overall decryption with the key, so we validate with a shorter 128 bit tag instead. The encryptObject method relies on authenticated encryption for the sake of

maintaining both integrity and confidentiality of the gradebook. We operate on AES as the symmetric key algorithm for its high computational security in combination with a Counter Mode block encryption mode for signing.

We initially thought of doing an external public and private key system, however, this would require the user to carry new keys after each add operation since the key is deterministic based on the encryption data. While appending the key directly into the encrypted gradebook object comes with its own risks, we rely on Java's security libraries to provide sufficient randomness of the 16 IV bytes as well as sufficiently secure incrementation of the IV for each block.

Decryption

Decryption occurs when the add and display programs load a new instance of the Gradebook struct into memory, with the decryption code being located within the struct itself. The IV is always located at the front of the encrypted byte array, so we take the first 16 bytes as the IV and the rest as our encrypted data. Java's cryptography libraries handle decryption from there, with an exception being thrown and handled in the event decryption fails for whatever reason, whether it be the checksum/counter mode decryption failing for integrity sake, or if the decrypted struct doesn't match the structure of a gradebook.

Potential vulnerabilities

- What if the key gets brute forced
  - AES256 is an industry standard algorithm and key length, with the brute force approach requiring millions of years and lots of energy to sustain the brute force, both of which your average person will not have access to.
- Direct modification of the encrypted object
  - Thanks to the counter mode encryption being used, both the key and the IV value are used to decrypt the serialized gradebook. In the event any of the surface-level data is modified from the gradebook, either the decryption will fail because the IV was corrupted or it will fail because the IV is used but decrypts and outputs trash, even if the key used is somehow correct.
  - Another byproduct of this serialization is additional obfuscation, making it harder for attackers to meaningfully modify the gradebook without explicitly seeing any data or knowing where data is placed within the struct
- Constant Initialization Vector placement, IV is directly exposed to attackers
  - The IV has a constant size and placement, attackers should be able to directly view the IV bytes and change them as they please, so that they can modify the encrypted data and change the IV to bypass integrity measures. We assume that the pseudorandom function provided by Java's standard library is random enough combined with a 16 byte tag to deter brute force, and even if the IV was correctly

reconstructed for the new data there's still a matter of the key. Combined with consideration of the previous vulnerability we have some good defense in depth measures.

- Denial of service through MitM
  - If attackers can't find any way to break the encrypted gradebook file, they could also just intercept the gradebook file and make sure the end user receives trash. Thanks to the peer to peer nature of the client and server setup, however, we can say that even if there was some man in the middle decimating gradebook files there will be some valid copy on either the server or client, and we can repeatedly have one or the other retry sending valid gradebooks.
- Inference deterrent
  - Any exceptions and errors related to decrypting the encrypted object or an invalid IV being provided are suppressed and the end user only receives an "invalid" message. This harms the user experience in the event they receive a corrupted gradebook and have no means to debug, but this scenario is an edge case worth discarding as the security provided with giving attackers less information on why decryption failed is far more valuable.

Member Contributions

- Jacob Riddle: Prepared an initially insecure version of gradebookadd and created test cases to validate more robust commands and potential failure cases.
- James Pham: Created an alternative gradebook struct that utilized a global assignment set and nested map for the purpose of a potential speed boost. Wrote the security implementation for encrypting and decrypting the gradebook.
- Kenny Song: Wrote and completed gradebook struct, display program, and added additional functionality to add program. Researched potential security implementations