****

# Practices for Secure Software Report

Table of Contents

[Document Revision History 3](#_Toc102040754)

[Client 3](#_Toc102040755)

[Instructions 3](#_Toc102040756)

[Developer 4](#_Toc102040757)

[1. Algorithm Cipher 4](#_Toc102040758)

[2. Certificate Generation 4](#_Toc102040759)

[3. Deploy Cipher 4](#_Toc102040760)

[4. Secure Communications 4](#_Toc102040761)

[5. Secondary Testing 4](#_Toc102040762)

[6. Functional Testing 4](#_Toc102040763)

[7. Summary 4](#_Toc102040764)

[8. Industry Standard Best Practices 4](#_Toc102040765)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **12/11/22** | **Carter Wooton** |  |

## Client



## Developer

Carter Wooton

## Algorithm Cipher

The algorithm cypher that I am choosing to use is the SHA-256, which is a successor to the SHA-1 family of algorithms. The purpose of the hashing function is to conceal data in a form that is illegible. This is useful for data such as passwords, where a database would store the hashed password rather than the original plaintext, or for checksums to verify that data is unchanged. The hashing algorithm takes plaintext as input and performs multiple mathematical operations on the bits, so the resulting string is a seemingly random sequence. The difference between hashing and encrypting is that hashing is not reversible: the original plaintext cannot be retrieved from the digest.

There have been many different ciphers used throughout the history of computing, and a common theme is that they begin life being very powerful, only to become obsolete and easily breakable in a decade or so. This is because computers become faster exponentially and can crack ciphers using brute force attacks. The bit level of the cipher describes how long the key is and how many rounds there are. The base 128-bit key for SHA-256 is strong enough to be un-breakable with today’s computers, but 196 and 256-bit keys exist for extra-sensitive communications. These higher bit levels require exponentially more computing power, so they may be unreasonable for most consumer software.

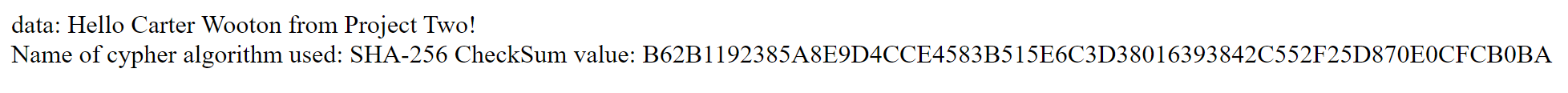
Symmetric keys are where both client and provider have one secret key that is used to both encrypt and decrypt messages; this encryption is best suited for data at rest and bulk encryption. Asymmetric keys are where the provider has a private key, and the client receives a public key; this encryption is suited for data in transit such as RESTful API interactions.

## Certificate Generation

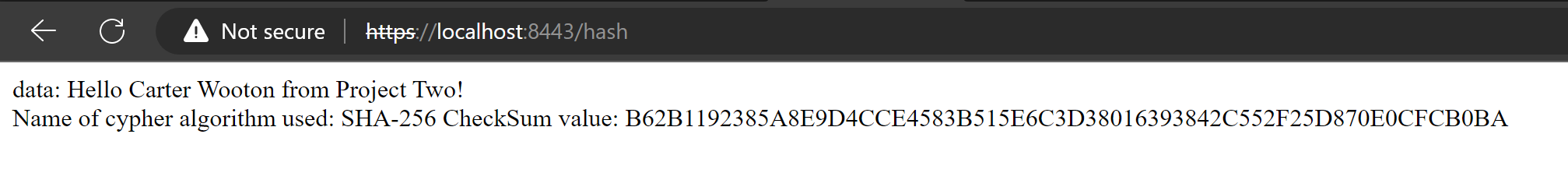
Text

Description automatically generated

## Deploy Cipher



## Secure Communications



Text

Description automatically generated

## Secondary Testing

Text

Description automatically generated

Code running without errors

Text

Description automatically generated

Before refactoring

Text

Description automatically generated

After refactoring

## Functional Testing

This program will eventually be verifying files to check that they are the original, unaltered files to ensure secure communications. Apart from the checksum, which is shown here in the refactored code, there will need to be file verification that checks that a file is the expected type and of the expected size.

## Summary

For this refactored code, I added a RestController class and added a Map for /hash, which for this case shows the checksum of secure communications using SHA-256. In the Application.properties file, I established the TLS connection to be used by this server by creating a self-signed certificate in my local keystore. This allows the HTTPS connection to be made, as opposed to just HTTP, which is unsecure. Specifically, the areas of vulnerability that I addressed by refactoring this code are cryptography, since we are dealing with encryption and hashing, and input validation, since the program will be working with file verification, which could include the uploading of files.

## Industry Standard Best Practices

I followed industry standard best practices when refactoring this code by first running an OWASP dependency check before changing the code, and then running another check after refactoring, and verifying that I did not introduce any more vulnerabilities into the code base. It is important that companies have a security-first mindset when developing a system, rather than leaving it till the end. This may add some weeks to the overall development process, but the payoff is in more robust systems that are not at severe risk of attack and will not cost the company millions of dollars or cause public distrust. In general, solving problems earlier in the SDLC will save the company money than solving them later.