****

# 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** | **April 17, 2024** | **Austin Scarinza** |  |

## Client



## Instructions

Submit these completed practices for secure software report. Replace the bracketed text with the relevant information. You must document your process for writing secure communications and refactoring code that complies with software security testing protocols.

* Respond to the steps outlined below and include your findings.
* Respond using your own words. You may also choose to include images or supporting materials. If you include them, make certain to insert them in all the relevant locations in the document.
* Refer to the Project Two Guidelines and Rubric for more detailed instructions about each section of the template.

## Developer

Austin Scarinza

## Algorithm Cipher

SHA-256, the Secure Hash Standard, produces a 256-bit hash value. The SHA-2 family is a well-known tool in cryptography and is considered very secure with no known vulnerabilities regarding collision attacks. SHA-256 and others in its family produce “a condensed representation of electronic data (message)” (Dang, 2012). With our goal of protecting files in transit, this hash function will properly condense the data on one side (sender) and then be un-hashed on the receiving end to verify it was received accurately. This hash function makes it extremely difficult to produce a different message that creates an identical hash value making the likelihood of there being a collision almost zero.

In the SHA-2 family, 128- and 192-bit lengths can also be used. Using SHA-256 is the most secure as the bit lengths are longer which means there are more possible combinations. SHA-256 is great for verifying data and ensuring that data has not been tampered with. Other encryption ciphers like AES work well in cases where data needs to be encrypted, transferred, and decrypted. This is often done on web applications and file storage. The use of symmetric like AES, asymmetric, and random numbers increase the changed security of the cipher. Performance also comes into play when choosing which type of cipher to go with.

Ciphers have been around for a long time as people have communicated. If people are talking, there is a need to share something in secret. Having secret code words or writing in a secret code such as Caesar cipher dates back thousands of years. As we are starting the transition from paper to computers, ciphers grew with the need to keep communication channels secure. As the people trying to crack ciphers become more intelligent on workarounds, the need for stronger more robust ciphers is inevitable.

## Certificate Generation

A screenshot of a computer program

Description automatically generated

A screenshot of a certificate

Description automatically generated

## Deploy Cipher

Insert a screenshot below of the checksum verification.

A screenshot of a computer

Description automatically generated

## Secure Communications

Insert a screenshot below of the web browser that shows a secure webpage.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Google and other web browsers still show self-signed certificates as not secure due to them being created on the server and not through a trusted CA. The certificate was added as shown in the screenshot but the site still shows as “not secure”.

## Secondary Testing

Insert screenshots below of the refactored code executed without errors and the dependency-check report.

A close-up of a text

Description automatically generated

A screenshot of a computer

Description automatically generated

## Functional Testing

Insert a screenshot below of the refactored code executed without errors.

A close-up of a text

Description automatically generated

## Summary

The code has been refactored by adding in a SHA-256 hashing algorithm to help create a checksum and hash data. This checksum is used to help the client and server verify that the data has not been tampered with. There was also the addition of a SSL certificate that will verify the server side and let users know it is a trusted site. The vulnerability Assessment Process Flow areas this helped with are API’s, cryptography, client/server, and code error.

The use of SHA-256 and MessageDigest is securing and using Api interactions. The application uses RESTful endpoints and adding extra layers of security to these points is important. Cryptography was expanded on a lot in the refactor. Before, there was no encryption or hashing, however, now there is some extra protection in place to protection data withing the application. As both sides of the application can see the certificate which is also double verified via the checksum, the client and server side both have some extra faith that both are operating as who they say they are. Lastly, the errors in place the state when the hashing function is not working correctly allow us to see and fix the code quickly if something were to happen.

## Industry Standard Best Practices

It’s important to make sure that when refactoring code, you don’t create more vulnerabilities than there were in the first place. It’s possible to make the code quicker or more efficient, but if more security vulnerabilities arise, then there is no real benefit, and the efficiency doesn’t matter as much. Steps in the right direction improve functionality while also improving, or at least staying consistent with the existing vulnerabilities.

I made sure not to introduce any new vulnerabilities while adding the hashing cipher to create the checksum and adding the certificate. I was able to verify this by running dependency checks throughout refactoring and making sure no new issues popped up. If they were to arise, I would look at how to fix them given fixes in the dependency check, and if I could not, I would have to evaluate the benefit of the added functionality against the drawback of the new vulnerability.

It is important that everyone takes this sort of though process when it comes to coding and adding to a company’s code base. Making small changes and verifying they are beneficial and still secure is better than large leaps forward that create security issues and eventually more work. If everyone had security in their mind while writing code, it would be a lot easier to check for a create fixes as development progresses. Following the best practices helps accomplish this and create secure code.

Dang, Q. H. (2012). Secure Hash Standard (SHS). NIST. https://www.nist.gov/publications/secure-hash-standard-shs?pub\_id=910977