****

# 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** | **06/18/2023** | **Robert Marlatt** |  |

## Client



## Developer

Robert Marlatt

## Algorithm Cipher

The best encryption algorithm cipher to use is AES since it accommodates a variety of key sizes and is one of the most significant standards currently in use. 128-bit or 256-bit encryption would be the most likely choice for this application, with 256-bit being extremely difficult for attackers to crack. The program will be able to deliver keys to its customers, who will be the recipients of communications, and encrypt data as necessary thanks to the development of symmetric keys.

The needs of the application determine whether symmetric or non-symmetric keys should be used. Public and private keys are included in non-symmetric keys, with the public key being known to all parties and the private key being known only to the client. The server and client both use the same symmetric keys. These keys are used to encrypt and decrypt data, and the right key must be used to unlock the data. One drawback is that if the encryption key for any encrypted data is lost, the data is likewise gone because it can't be decoded without the key. It is possible to employ random number generators to give transactions a distinctive identity, which can help with identifying certain activities like data transfers or discussions.

In short, AES uses up to 256 bits to encrypt data, making it virtually hard to decrypt owing to the wide range of potential values. To assure the security of communication between the server and the client for the purposes of this application, we employed 128-bit encryption.

## Certificate Generation

Insert a screenshot below of the CER file.

A screenshot of a computer

Description automatically generated

## Deploy Cipher

Insert a screenshot below of the checksum verification.

A screenshot of a computer

Description automatically generated with medium confidence

## Secure Communications

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

A screenshot of a computer

Description automatically generated with medium confidence

Chrome will not allow a self signed certificate to be considered secure. Could not show a secure site.

## Secondary Testing

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

A picture containing text, screenshot, software, display

Description automatically generatedA screenshot of a computer

Description automatically generated

## Functional Testing

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

A screenshot of a computer program

Description automatically generated with medium confidence

## Summary

The "CS 305 Project Two Code Base.zip" attachment contains the refactored code. We solved the following security issues by reworking the code: APIs, cryptography, client/server, and code quality. We produced keys for this program and our own self-signed certificate, enabling us to connect using 256-bit AES encryption. The data is encrypted so that only the intended recipients can read it, ensuring the security of the application communications. In this application, it is crucial to maintain secure connections since unsecure communications might allow an attacker to view data and steal sensitive data. A security breach can result in a loss of trust between the owner of the program and the user, financial loss from government fines for not requiring safe communications, and financial loss from resources needed to fix or resolve the security issue. By keeping the application secure, the organization can safeguard its goods and resources while retaining the confidence and business of its customers.

## Industry Standard Best Practices

The best practices for maintaining current security include checking the code periodically for vulnerabilities, particularly after adding new features or altering existing ones and before publishing or applying the changes in the live code. The development team should focus on patching any newly discovered vulnerabilities if they are discovered during the creation of new or updated functionality (before they go live). They should choose the appropriate course of action and ascertain whether any application-using parts are affected by any vulnerabilities for which there are currently no remedies. The team can omit those vulnerabilities from their reports if they discover that they exist for any dependencies utilized but have no impact on any code-using parts. The development team will need to keep an eye out for upgrades or fixes for any vulnerabilities that are already present in the code.