Software Requirements Specification

for

Secure Module/Key Management Application

Version 2.3 approved

Prepared by Andy Munoz, Anthony Ortega, Alexander Raya, Angel Penate, Angel Serrano, Brian Tang, Victor Liang, Jorge Espana, Gabriel Espejel, and Jason Schmidt

QTC

October 11, 2023

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Victor Liang	10/18/23, 10/25/23	Introduction, Purpose, Intended Audience, Product Scope, Overall Description, System Analysis	1.0 - 1.1
Jason Schmidt	10/18/23, 10/25/23	User Classes and Characteristics, User Interfaces, Software Interfaces, Functional Requirements, Logical Database Requirements	1.0 - 1.1
Brian Tang	10/18/23, 10/25/23	Operating environment, User documentation, Legal and Ethical Considerations, Appendix A - C, Hardware interfaces	1.0 - 1.1
Angel Serrano	10/25/23	Security Requirements	1.1
Jason Schmidt	11/1/23	External Interface Requirements	1.2
Angel Serrano	11/1/23	Software Quality Attributes & Business Roles	1.2
Brian Tang	11/7/23	Definitions, Acronyms, and Abbreviations Communications Interface	1.3
Anthony Ortega	11/7/23	Purpose	1.3
Jason Schmidt	11/8/23	Product Perspective, Design and Implementation Constraints	1.4
Jason Schmidt	11/15/23	System Analysis, Intended Audience and Reading Suggestions	1.5
Anthony Ortega	11/15/23	Performance Requirements	1.5
Anthony Ortega	11/15/23	Functional Requirements	1.5
Angel Serrano	12/6/23	Product Functions	1.6
Angel Serrano	4/12/24	Analysis Model Update and Use Case Steps Update	2.1
Brian Tang	4/14/24	Use Case 1, 2 & 3 diagrams updated	2.1
Jason Schmidt	4/17/24	Updated Glossary and made some changes to Design Constraints	2.2
Andy Munoz	4/17/24	Updated implementation changes and made major changes to Design Constraints	2.2
Victor Liang	4/23/24	Added Definitions	2.3

1. Introduction

Secure Module/Key Management Project is an application that does secure data transfer between two points in three use cases: passing data between servers in a trusted domain, passing data from a server in a trusted domain to another server outside the trusted domain, passing data from a server in a trusted domain to another server outside the trusted domain over an unknown number of intermediary systems. The application will use HTTPS or SSL to transfer data. There is a logging system that logs all HTTP activity. The logged data should be encrypted and visible to users. The application will use Microsoft Azure Vault, ASP.NET Core, and Microsoft SQL Server. This application will also use certificates to authenticate and provide a secure data transfer connection.

1.1 Purpose

The purpose of this document is to explain what the Secure Module/Key Management Application does and how it meets QTC requirements. Additionally, this document covers the core aspects of the software that define the purpose of building it. We will go over the internal and external scope of the software as well as its general characteristics.

In order to verify and validate the requirements of the Secure Module/Key Management Application and provide QTC with a robust and reusable software, this document will detail test strategies that may be implemented for future software technicians throughout the entire development cycle for key life-cycle management and data security throughout trusted and untrusted domains.

1.2 Intended Audience and Reading Suggestions

This document is intended for Software Engineers, Operational Users, and other roles added in the future.

Software Engineers may refer to the overall description of the document to understand its classes, operations, and documentation.

Operational Users and Software Engineers are encouraged to have a high-level understanding of the application by reading sections 1, 2, and 3.

1.3 Product Scope

The Secure Module/Key Management Application will securely transfer files between two points in three different use cases and store encrypted information in a database. This application has key management and key rotation when needed using Azure Key Vault. Those with required access will see only the encrypted data, while others with the correct permissions will see unencrypted data.

Goals:

1. Passing data from one server to another inside a trusted domain across an HTTPS connection, but HTTP activity is logged to a central logging database that users can access. Sensitive data (PHI/PII) that is part of the data payload must be protected from being visible to users accessing the logs.
2. Passing data from one server in a trusted domain to another outside the trusted domain across an HTTPS connection. HTTP activity is logged as detailed above.
3. Passing data from one server in a trusted domain to another server outside the trusted domain with an unknown number of intermediary systems handling the data between the two endpoints. Data is transmitted across HTTPS connections, and HTTP activity should expect to be logged as detailed above.

1.4 Definitions, Acronyms, and Abbreviations

Secure Module - a distinct hardware or software component of a more extensive system or application meticulously planned and executed with security in mind. The purpose of secure modules is to offer an extensive degree of safeguarding against a wide range of security threats, vulnerabilities, and assaults.

Key Management - systematic operations and protocols encompassing the generation, storage, distribution, and control of cryptographic keys employed to safeguard data, communications, and resource access within computer security and cryptography. A critical element in ensuring secure data protection and communication is the management of keys.

HTTPS - an acronym for Hypertext Transfer Protocol Secure, is a secure communication extension that augments the standard HTTP protocol when employed across a computer network, including the Internet. Its purpose is to bolster the security and credibility of the information transmitted between a web server and a user's browser.

Key Vault Certificates - are cryptographic credentials employed for many security and encryption objectives across multiple systems, including Azure.

Microsoft:

- Azure Key Vault - a cloud-based service that enables organizations to store and administer sensitive data, including cryptographic keys, secrets, and certificates, in a secure manner.

- SQL Server - a system or software application that utilizes the Structured Query Language (SQL) to store, administer, and retrieve data from relational databases. A SQL Server is a database management system (DBMS) that ensures the establishment and upkeep of relational databases. SQL is a specialized language utilized to manage and query structured data.

ASP.NET Core - a cross-platform, open-source framework for developing web applications created by Microsoft. It is a contemporary, modular framework for developing microservices, APIs, and web applications. ASP.NET Core represents a substantial progression from its predecessor, the ASP.NET framework, and provides many significant benefits, such as enhanced productivity for developers, improved performance, and cross-platform compatibility.

Key Rotation/Rolling - periodically update cryptographic keys to enhance security. This plays a crucial role in maintaining the security of applications and data.

RSA - A type of asymmetric encryption that utilizes a public and private key. The receiver's public key is shared and used to encrypt information and sent to the receiver that can only be encrypted with the receiver's private key. So information transferred using RSA is safe but is slow to encrypt and decrypt.

AES(Advanced Encryption Standard) - A symmetric encryption algorithm that utilizes a key to quickly encrypt and decrypt data.

PHI/PII(Personally Identifiable Information) - Sensitive information that can identify a person such as Social Security number, name, date of birth.

SDK(Software Development Kit) - Provides libraries

1.5 References

- Get started with Key Vault certificates | Microsoft Learn
  - https://learn.microsoft.com/en-us/azure/key-vault/certificates/certificate-scenarios
- Certificate creation methods | Microsoft Learn
  - https://learn.microsoft.com/en-us/azure/key-vault/certificates/create-certificate
- Using HashiCorp Vault C# client with .NET Core
  - https://developer.hashicorp.com/vault/tutorials/app-integration/dotnet-httpclient
- Azure encryption overview | Microsoft Learn
  - https://learn.microsoft.com/en-us/azure/security/fundamentals/encryption-overview
- Encrypt and Decrypt Data with C# and SQL Server [closed]
  - https://stackoverflow.com/questions/7921943/encrypt-and-decrypt-data-with-c-sharp-and-sql-server

2. Overall Description

This application will offer a secure way to transfer and store sensitive information. This application will securely transfer information and store encrypted information in a database. This application will apply key management and rotation to change encryption keys within a time period.

2.1 System Analysis

Goals:

1. Passing data from one server to another inside a trusted domain across an HTTPS connection, but HTTP activity is logged to a central logging database that users can access. Sensitive data (PHI/PII) that is part of the data payload must be protected from being visible to users accessing the logs
2. Passing data from one server in a trusted domain to another outside the trusted domain across an HTTPS connection. HTTP activity is logged as detailed above.
3. Passing data from one server in a trusted domain to another server outside the trusted domain with an unknown number of intermediary systems handling the data between the two endpoints. Data is transmitted across HTTPS connections, and HTTP activity should expect to be logged as detailed above.

Major Technical Hurdles:

1. Encryption
2. Decryption
3. Rotating Keys

Solutions:

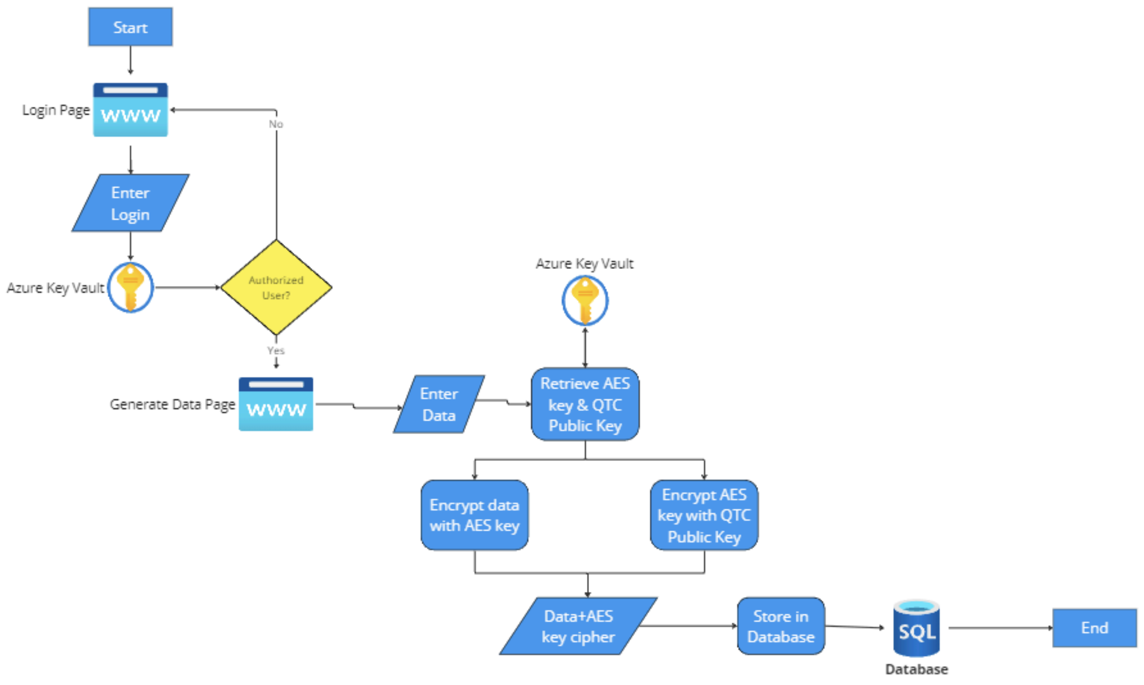
1. The application will use built-in C# functions for encryption
2. The application will use built-in C# functions for decryption
3. The application will measure time and rotate keys after an appropriate and predetermined amount of time.

2.2 Product Perspective

QTC will utilize this software to store and retrieve sensitive data while protecting it from potential leaking of that data in any intermediate steps. As such, this software will be connected to a database and key vault. Namely, the Secure Module/Key Management Application will utilize Microsoft SQL Server and Azure Key Vault. The software is based on the idea of key life cycle management. KLCM forms the standard for all modern data security and data transfer. To implement this management solution, we will be using Azure Vault. Azure Vault guards cryptographic keys used in the cloud applications and services.

2.3 Product Functions

Use Case 1



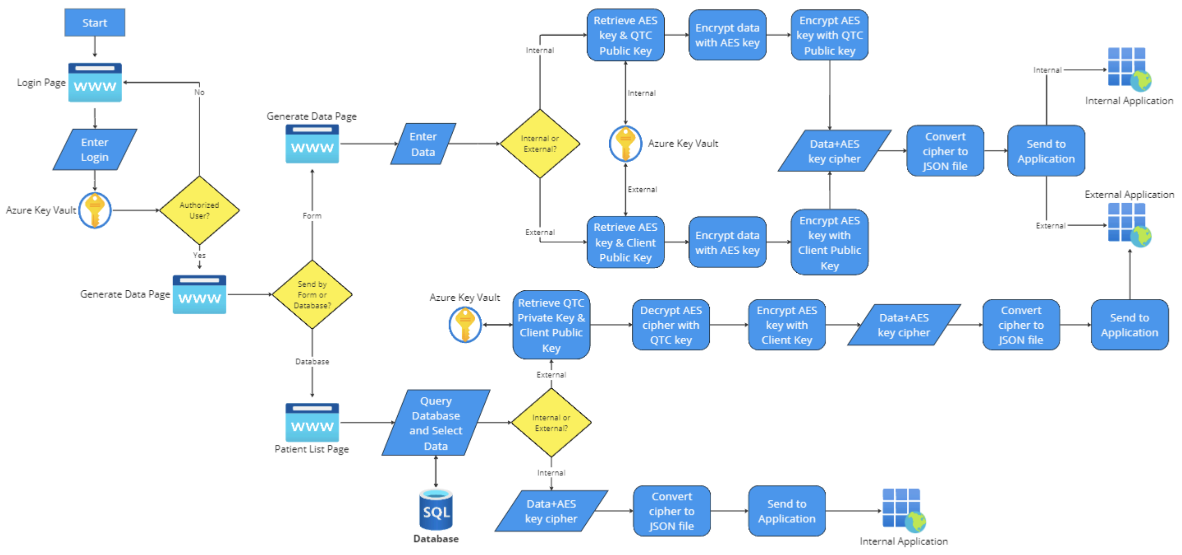
The application initiates a secure login page, where users are required to authenticate themselves by entering their credentials. Upon successful verification, users are directed to the main

"Generate Data" page. In cases where authentication fails, users are redirected back to the login page, accompanied by an error message.

The user can then enter data into the provided form. Following this, the system retrieves an available AES key along with QTC's public RSA key from Azure Key Vault. The entered data is then encrypted using the AES key. To further secure the data, the AES key itself is encrypted using the public RSA key. Finally, both the encrypted data and the encrypted AES key are securely stored in the database. This process ensures that the data remains protected at every stage of handling.

This diagram is designed exclusively for internal use at QTC and is built on a foundation of trust and security. It incorporates a sophisticated mechanism to allow QTC's application access to the vault to retrieve the decryption key, enabling secure data decryption when necessary. The use of AES encryption highlights the use of standard and common encryption practices.

Use Cases 2 and 3


Start
Login Page
Enter Login
Azure Key Vault
Authorized User?
No
Yes
Generate Data Page
Send by Form or Database?
Form
Database
Generate Data Page
Enter Data
Internal or External?
Internal
External
Retrieve AES key & QTC Public Key
Encrypt data with AES key
Encrypt AES key with QTC Public key
Azure Key Vault
Retrieve AES key & Client Public Key
Encrypt data with AES key
Encrypt AES key with Client Public Key
Data+AES key cipher
Convert cipher to JSON file
Send to Application
Internal Application
External Application
Patient List Page
Query Database and Select Data
SQL
Database
Retrieve QTC Private Key & Client Public Key
Decrypt AES cipher with QTC key
Encrypt AES key with Client Key
Data+AES key cipher
Convert cipher to JSON file
Send to Application
Data+AES key cipher
Convert cipher to JSON file
Send to Application
Internal Application

The application commences with a robustly secure login interface, requiring users to authenticate by providing their credentials. Once the credentials are successfully verified by the vault, users are navigated to the "Generate Data" page, the central hub of the application. If the authentication process fails, the user is redirected back to the login page with an informative error message.

Users can enter data into a form. Depending on the destination of the data, the process diverges: for internal applications (Use Case 2), the company's public RSA key is retrieved from Azure Key Vault; for external applications (Use Case 3), the third party's public RSA key is instead retrieved. The data entered is encrypted using an AES key obtained from Azure Key Vault, and this AES key is then encrypted with the appropriate public RSA key. The encrypted data and AES key are converted into a JSON file, which is sent to the internal application.

For data being sent from the database, the user again logs in and authenticates, accessing the same Home Page and then navigating to the View Patients Page. After entering a search query and selecting the data, if it's destined for an internal application, the data remains unmodified. However, if it's for an external application, the company's private key is used to decrypt the AES key. Then, after retrieving the third party's public RSA key, the AES key is re-encrypted. The encrypted data and key are converted into a JSON file, and certificates are exchanged via Azure Key Vault. The authentication of the third party then determines the data exchange: if authenticated, the encrypted data is exchanged and authenticated; if not authorized, the exchange does not occur. This process ensures secure data handling and transfer tailored to the specific requirements of the recipient.

A key aspect is the protocol for third-party access, mandating that third parties authenticate themselves via Azure to access decryption keys. These keys are designed to expire after each use, bolstering security dynamically. Additionally, SSL Certificates are meticulously managed, tracked, and renewed to maintain a continuous state of secure communication. The SSL Certificates portion was not implemented as it was not a requirement.

2.4 User Classes and Characteristics

- Operational User - creates, encrypts, and views encrypted/decrypted data
- Data Technician - can view encrypted data from HTTP logs from a separate database

2.5 Operating Environment

Our software will be based on Windows OS. Another software component is Microsoft Azure, which we will use to create and implement key vault certificates. The primary coding language our team will use to create our software is C# (C Sharp).

2.6 Design and Implementation Constraints

- Interfaces with other applications
    - Microsoft SQL Server
    - Azure Key Vault
- Reliability requirements
    - As it is meant to be secure, the software will be failure-free for its specified environment.
- Safety and security considerations.
    - The software must encrypt data with the appropriate encryption method
    - The software must only send encrypted data
    - The software must only store encrypted data

2.7 User Documentation

- Get started with Key Vault certificates | Microsoft Learn
    - https://learn.microsoft.com/en-us/azure/key-vault/certificates/certificate-scenarios
- Certificate creation methods | Microsoft Learn
    - https://learn.microsoft.com/en-us/azure/key-vault/certificates/create-certificate
- Using HashiCorp Vault C# client with .NET Core
    - https://developer.hashicorp.com/vault/tutorials/app-integration/dotnet-httpclient
- Azure encryption overview | Microsoft Learn

    - https://learn.microsoft.com/en-us/azure/security/fundamentals/encryption-overview
- Encrypt and Decrypt Data with C# and SQL Server [closed]
    - https://stackoverflow.com/questions/7921943/encrypt-and-decrypt-data-with-c-sharp-and-sql-server

2.8 Assumptions and Dependencies

It is assumed that most operating systems and browsers will work with the application.

- Microsoft SQL Server
- Microsoft Azure Key Vault
- ASP.NET

2.9 Apportioning of Requirements

Performance Requirements:

- Response Time: The web application should respond to user input within 2 seconds under normal operating conditions.
- Throughput: Not specified.
- Data Processing: The application should encrypt and decrypt data within 1 second for any individual operation to ensure a seamless user experience.
- Uptime/Availability: The service should be available 99.9% of the time, with downtime only occurring during planned maintenance windows.

Scalability Requirements:

- User Load: Not specified.
- Data Volume: The database should be capable of scaling to accommodate increased data load.
- Resource Utilization: The application should optimize resource use and maintain responsiveness even as the number of requests increases.
- Deployment: The application can support deployment in a cloud environment that automatically scales resources based on demand (e.g., Azure App Service or Kubernetes).

Traceability Requirements:

- User Actions: Not specified.
- Data Access and Changes: Application itself only has access and not users.
- System Changes: Not specified or tracked.
- Error Tracking: Not specified or tracked.

3. External Interface Requirements

3.1 User Interfaces

This project is a web application where users can log in and send/receive data. This means there needs to be an interface allowing users to input identifying information that the system could use to log them into their account (i.e., username/email and password). However, this is not implemented as this is a proof of concept for QTC.

To comply with the Americans with Disabilities Act, the application will be made with accessibility in mind. This means using fonts and font sizes that are easily readable and selecting colors that contrast nicely and do not blend together. Moreover, the more interactive portions of the interface (i.e., buttons, text boxes, etc.) will be clearly labeled and purposefully stand out.

3.2 Hardware Interfaces

There will be no hardware interaction since we will build a web application. That means it will be software-based. Our software will be very straightforward to run on. Running our software on a working computer is only the bare minimum since it will not be working hard on tasks. We will send data forward and back for the backend, but that will rely on the server resources, not our computer specs. Overall, our front end will be talking to the browser, which is our web application, and the browser relays the information to the backend. That is how the front and backend will be communicating with each other within the browser.

3.3 Software Interfaces

- Microsoft SQL Server
    - Microsoft SQL Server 2022
    - Microsoft SQL Server will interface with our system by storing encrypted data for later use by our application.
- Azure Key Vault
    - Make HTTP requests to the Key Vault REST API endpoint, typically "https://<your-key-vault-name>.vault.azure.net/"

3.4 Communications Interfaces

For our communications interface, we will use HTTPS to communicate throughout the network (backend). Which would interact with running web services or APIs. HTTP or HTTPS will be in charge of receiving responses and making API requests (communications). As for security measures, enabling HTTPS for secure communications, authentication, and handling API tokens. We must also optimize API requests for efficiency and responsiveness for faster data transfer rates.

4. Requirements Specification

4.1 Functional Requirements

- The program will:
    - generate data
    - encrypt the data
    - save the encrypted data to the database
    - send the encrypted data to an authorized application
    - validate credentials to allow users to log in and access data
    - rotate keys used for encryption after a set amount of time
    - associate keys respectively
    - data fields to accept a string of up to 250 ASCII characters

4.2 External Interface Requirements

- Login Data (Input)
    - This is the identifying information used to log in the correct user:
        - Username/Email
            - Retrieved by having a user input their data into a text field
        - Password
            - Retrieved by having a user input their data into a text field.
- Encrypted Data (Output)
    - This is the data the program will send to the internal or external application:
        - The program needs to send the encrypted data to the internal or external application, who will then decrypt it for viewing. To achieve this, the program will first have to generate data, then encrypt it, and lastly, send the encrypted data.

4.3 Logical Database Requirements

- The database:
    - will be used every time data is encrypted and needs to be stored
    - will use sensitive data types (i.e. SSN, name, email, etc.)
    - cannot be viewed or accessed directly by any users
    - will keep user-sensitive information for as long as its stated purpose, healthcare information may be stored for decades

    - will enforce several integrity checks, i.e. database will follow the relational data model (ensure entity, referential, and domain integrity are kept intact)
- For provided data models/diagrams, check Appendix B: Analysis Models.

4.4 Design Constraints

1. Platform Compatibility:

- The application is compatible with multiple operating systems, including Windows, Mac, and Linux. However, Mac users will require an alternative development environment as Visual Studio is no longer supported.
- The web app is optimized for a range of browsers, including Chrome, Firefox, Edge, and Safari.

2. Connectivity:

- Continuous internet connection is necessary for accessing the database and Azure services like Key Vault. Offline capabilities are not supported.

3. Security and Privacy:

- All sensitive data, including personal patient information, will be encrypted using AES and RSA algorithms, with keys managed by Azure Key Vault to ensure confidentiality and compliance with data protection regulations.
- Authentication and authorization of the application itself are managed via Microsoft Entra, ensuring secure and controlled access to application features.

4. Data Storage and Management:

- The database schema supports the necessary fields for patient data, and any modifications should maintain data integrity and support future scalability.

5. User Experience:

- The user interface is intuitive and accessible.

6. Performance:

- The application is optimized for load times and efficient database queries to handle high traffic without performance degradation.
- Server and client-side processing is optimized to ensure a responsive user experience.

7. Maintainability:

- The codebase follows industry-standard practices for readability, extensibility, and maintainability, with comprehensive documentation for future developers.
- We used GitHub for version control for tracking changes and collaborative development.

8. Compliance and Regulations:

- Our application doesn’t fully comply with HIPAA (Health Insurance Portability and Accountability Act) standards in the U.S. as this application is mainly a proof of concept and not meant for production use.

9. Integration:

- The web app is not designed to integrate with other systems and APIs, both internal and external, without compromising security or performance due to the application being mainly a proof of concept and not meant for production use.

10. Infrastructure:

- The backend infrastructure is cloud-based and supports scalability, disaster recovery, and high availability. However, not implemented due to the application being mainly a proof of concept and not meant for production use.
- Cloud resource utilization can be monitored and optimized to control costs without sacrificing performance.

5 Other Nonfunctional Requirements

5.1 Performance Requirements

The number of terminals accessing the data will be dynamic as multiple clients may access the data they sent. The amount of data and type will be dynamic as we must be able to encrypt all data the client may need to encrypt and send to our application. 100% of all data processed will be stored on the database and then accessed by our application to retrieve the data.

5.2 Safety Requirements

Keys will be generated for the client to access the data they encrypt and be guarded inside the key vault. The Key Vault will keep track of any logging and access of transferred, encrypted, and decrypted data.

5.3 Security Requirements

Azure Key Vault will store and maintain credentials, keys, certificates, and registered applications. The keys will only be given to authorized the application itself for decryption. The software security will be entirely based upon the Principle of Least Privilege, which means the user or entity shall only have access to specific data, resources, and applications to complete a necessary task.

5.4 Software Quality Attributes

The data, once inputted, will be encrypted, stored, and sent to its destination, and in each, all data can be decrypted with the apt key given the certified key holders from the key vault. Data sent by HTTPS will be securely transferred by SSL but encrypted with RSA, which provides ample security for the type of transferred data, as no one between the client and server should be able to read the data. Utilizes SSL Certificates to verify 3rd Party Application.

5.5 Business Rules

Our application can encrypt, decrypt, send, and store data within QTC. Data technicians will have no access to the data. They can only see the traffic logged from QTC.

6. Legal and Ethical Considerations

- Data Privacy Laws and Compliance: The application must comply with local and international data privacy laws, such as the General Data Protection Regulation (GDPR) in Europe, the California Consumer Privacy Act (CCPA) in the USA, and other relevant regulations. This includes obtaining consent for data collection, ensuring the right to access and delete personal information, and reporting data breaches in a timely manner.
- Encryption Standards: While using AES and RSA encryption provides a high level of security, it's essential to ensure that implementing these encryption methods complies with industry standards and best practices. Inadequate or improper implementation can lead to vulnerabilities.
- Data Access and Authentication: The mechanisms for user authentication and data access must be robust and regularly updated to prevent unauthorized access. This includes ensuring that third-party access, as mentioned in your use cases, is securely managed and monitored.
- Secure Data Transmission: SSL certificates for secure data transmission are crucial. However, it's important to ensure these certificates are properly managed, updated, and protected against unauthorized access or misuse.
- Key Management: The management of encryption and decryption keys is a critical aspect. This includes secure storage, access control, and regular rotation or expiration of keys to minimize risks.
- Third-Party Vetting and Compliance: Any third-party applications or services involved in data transactions must be thoroughly vetted for compliance with security standards and legal requirements. This is especially important when handling sensitive or personal data.
- Ethical Use of Data: Beyond legal requirements, there's an ethical obligation to use data responsibly. This includes transparency about how data is used, stored, and shared, as

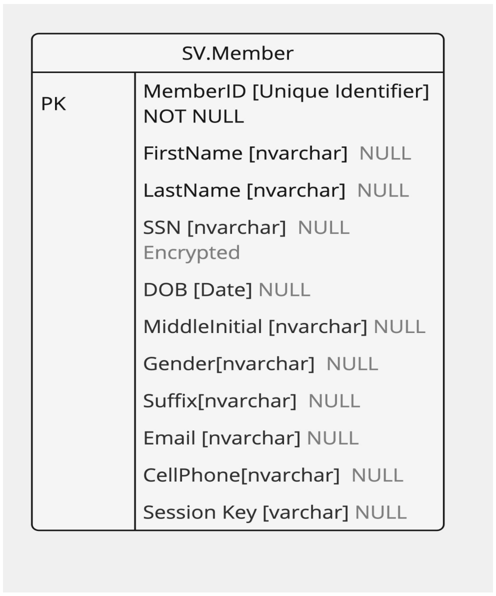
well as ensuring that the data is not used for purposes that could be considered unethical or harmful.

- Audit Trails and Monitoring: Regular audits and continuous monitoring of the system are important to ensure compliance with legal and ethical standards, as well as to identify and rectify any security vulnerabilities promptly.
- Incident Response Plan: Having a robust incident response plan in case of data breaches or security incidents is legally and ethically important. This includes procedures for notification, mitigation, and post-incident analysis.
- User Awareness and Training: Ensuring that users are aware of the security protocols and understand their role in maintaining data security is also crucial. This might involve regular training sessions and updates on security practices.

Appendix A: Glossary

- PHI/PII - Personally Identifiable Information.
- RSA - Rivest–Shamir–Adleman.
- AES - Advanced Encryption Standard.
- HTTPS - Hypertext Transfer Protocol Secure
- MS SQL - Microsoft SQL Server
- ASP.NET - Active Server Pages Network Enabled Technologies
- SDK - Software Development Kit

Appendix B: Analysis Models



SV.Member	
PK	MemberID [Unique Identifier] NOT NULL
	FirstName [nvarchar] NULL
	LastName [nvarchar] NULL
	SSN [nvarchar] NULL Encrypted
	DOB [Date] NULL
	MiddleInitial [nvarchar] NULL
	Gender[nvarchar] NULL
	Suffix[nvarchar] NULL
	Email [nvarchar] NULL
	CellPhone[nvarchar] NULL
	Session Key [varchar] NULL

Appendix C: To Be Determined List

N/A

Senior Design Final Report

Secure Module/Key Management Application

QTC® Quality. Timeliness. Customer Service.

A Leidos Company

Version 1.0 - 05/09/2024

Team Members:
Andy Munoz
Angel Penate
Alexander Raya
Gabriel Espejel
Brian Tang
Victor Liang
Anthony Ortega
Jorge Espana
Angel Serrano
Jason Schmidt

Faculty Advisor:
Dr. Huiping Guo

Liaisons:
Francisco Guzman
Julian Gutierrez

Table of Contents

1. Introduction:

1.1. Background:

QTC Management, acquired by Leidos in 2016, is at the forefront of providing specialized medical examination and diagnostic services, with a focus on disability assessments. QTC helps manage and store the sensitive data of many people, a task that is not easily accomplished. Therefore, QTC has entrusted us to develop a Secure Module/Key Management Project that performs secure data transfer between two points in three use cases: passing data between servers in a trusted domain, passing data from a server in a trusted domain to another server outside the trusted domain, passing data from a server in a trusted domain to another server outside the trusted domain over an unknown number of intermediary systems. The application will use HTTPS or SSL to transfer data. There is a logging system that logs all HTTP activity. The logged data should be encrypted and visible to users. The application will use Microsoft Azure Vault, ASP.NET Core, and Microsoft SQL Server. This application will also use certificates to authenticate and provide a secure data transfer connection.

1.2. Design Principles:

The application is the main deliverable, and the app uses various other technologies, applications, and frameworks to accomplish its goal of securely storing and sending patient's data. Additionally, since the program is not intended for use only by people who understand the entire structure of the application backend (such as the underlying code) it has been equipped with an easy-to-understand user interface, alongside input validation. Thus, the user can easily correct mistakes they made while inputting data. Moreover, since the main goal of the application is to securely store and transfer patient data we needed to make sure the program used effective encryption/decryption techniques. To ensure data was kept secure we utilized a combination of AES (a symmetric key algorithm) and RSA (an asymmetric key algorithm). Lastly, we implemented a key rotation algorithm into our system using Quartz.net to ensure keys were never used for too long, adding layers of security.

1.3. Design Benefits:

By having an application that can scale with any amount of patients we have made it much easier for our project sponsor, QTC, to utilize our application. Moreover, since our project utilizes the MVC architecture and uses pre-existing technologies it is much easier for QTC to expand upon the application in the future. The design of the Secure Module/Key Management Application incorporates several key benefits that enhance both security and user experience. Firstly, the use of Microsoft Azure Key Vault at the core of the key management strategy provides robust security measures by centrally managing cryptographic keys and secrets, which are essential for data encryption processes. This integration not only simplifies key management but also significantly reduces the risk of unauthorized access. Additionally, the application's modular architecture allows for scalability and flexibility, facilitating future enhancements and integration with other systems or technologies without disrupting existing functionalities. The intuitive user interface is designed to ensure that users can efficiently manage their tasks with minimal training, promoting better user engagement and productivity. Overall, the strategic design choices made in developing this application result in a secure, scalable, and user-friendly platform that supports the complex needs of secure data management and compliance with industry standards.

1.4. Achievements:

1. Technical Proficiency

- Applied .NET Framework, Azure Key Vault/Database for our Secure Module/Key Management Application
- Overcame challenges using encryption/decryption techniques
- Innovative features and solutions developed as part of our application

2. Team Collaboration & Workflow Optimization

- Adapted/improved the use of Git/Gitchub version control
- Organized our branches in specific categories for the success of our branch control strategies that helped manage our project more efficiently
- Met our deadlines and improved the quality of our code with coordinated code sprints

3. Professional Development

- Better note-taking and more informative commits have enhanced project transparency and troubleshooting efficiency
- Preemptive merge resolution prevented potential conflicts and streamlined our development process

2. Related Technologies:

Results and Interesting Findings

- The implementation of the Secure Module/Key Management Application successfully achieved its primary goal of securely transferring data between servers within and outside the trusted domain, utilizing HTTPS for secure transmission and Microsoft Azure Vault for key management.
- The integration of Azure Key Vault significantly enhanced the security framework by securely managing and rotating cryptographic keys, demonstrating a robust mechanism against potential security breaches.
- However, the system faced challenges in achieving seamless integration with some third-party services, particularly in scenarios involving numerous intermediary systems where maintaining the integrity and security of the data became more complex.

Successes:

- The application successfully encrypted and decrypted data transfers between servers, ensuring that sensitive data (PHI/PII) remained protected during transit and at rest.
- The use of AES and RSA encryption methods provided strong security measures that met industry standards for data protection.
- The logging mechanism was effective in capturing HTTP activity without exposing sensitive data, adhering to privacy and security regulations.

Failures:

- The application did not fully comply with HIPAA standards, which limited its deployment in environments requiring strict compliance with health data protection laws.
- Some issues were encountered with the user interface, particularly in terms of accessibility and ease of use, which could hinder user adoption and overall satisfaction.

Follow-up Research and Development:

- Compliance Enhancement: Further research into adapting the system to fully meet HIPAA and other relevant data protection standards is necessary. This will likely involve a thorough review of encryption methods and audit controls.
- User Interface Improvement: Development efforts should focus on enhancing the user interface, making it more intuitive and accessible to ensure it meets the diverse needs of all users, including those with disabilities.
- Third-party Integration: Continued development is needed to improve integration capabilities with external systems, especially in complex scenarios involving multiple intermediaries. This might include the development of more robust error handling and data integrity checks.

Performance Optimization:

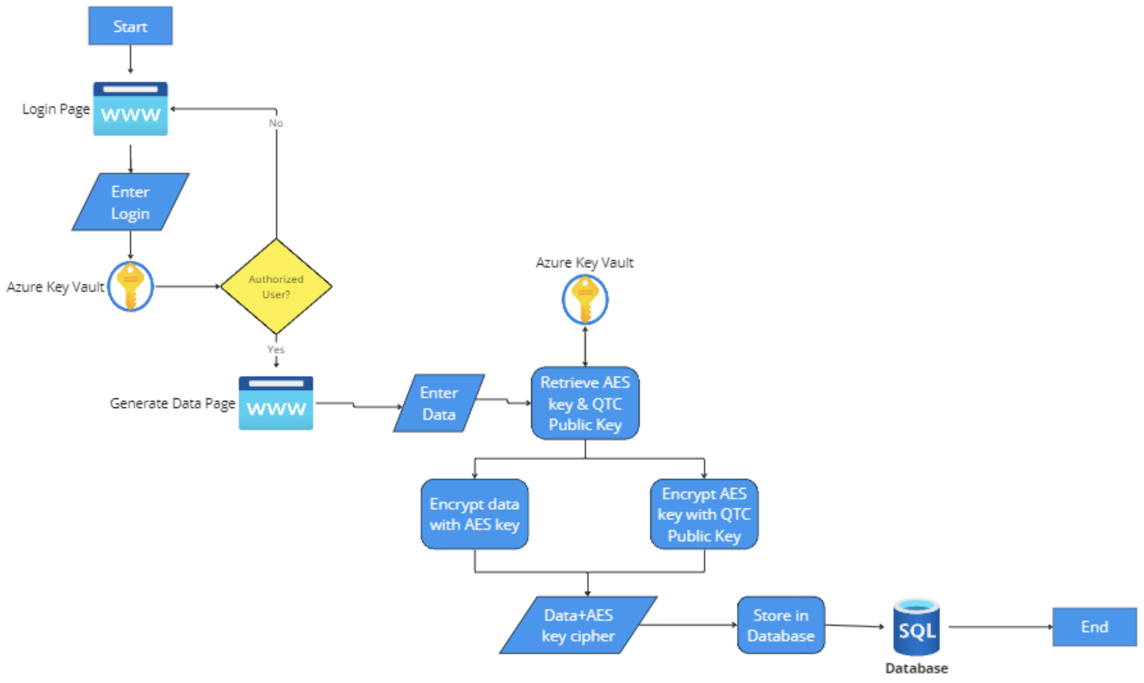
- Ongoing optimization of the system's performance under various loads is crucial, particularly focusing on reducing latency and increasing throughput for high-volume data transfers.

3. System architecture:

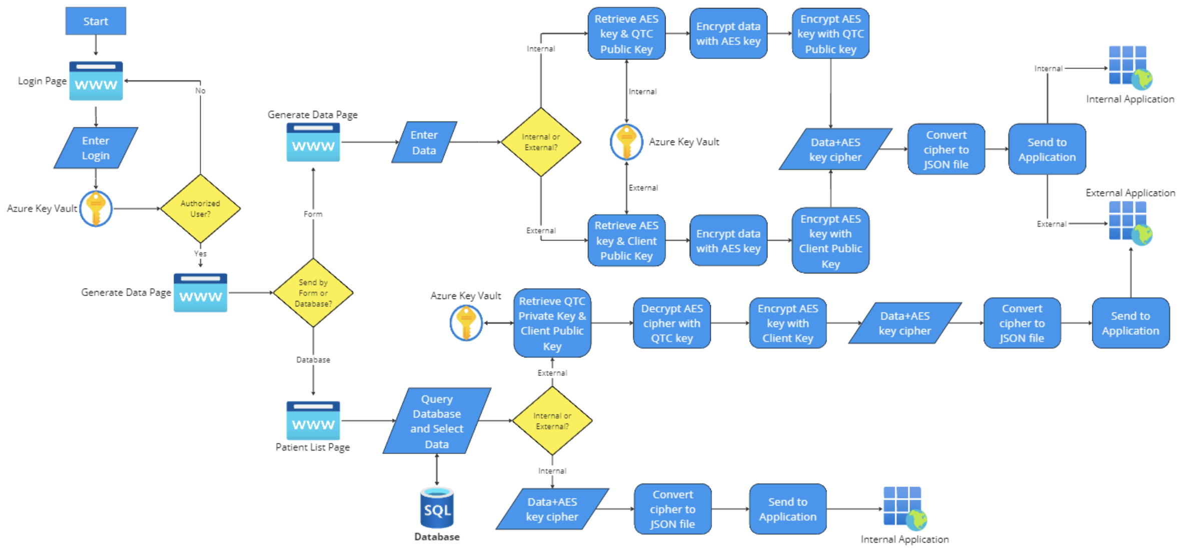
3.1. Overview:

The architecture for the web application.

Here is a diagram that shows how this architecture works:



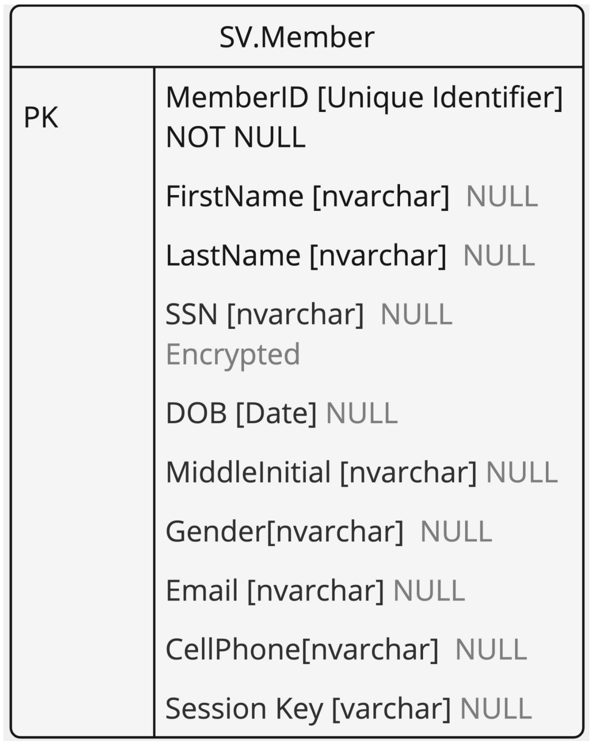
Use Case 1: The process involves the secure transmission of data containing sensitive information (such as Personal Health Information [PHI] and Personally Identifiable Information [PII]) between servers within a trusted domain. The communication channel is safeguarded using HTTPS to ensure encryption in transit. Concurrently, HTTP requests and responses are logged for auditing or analytical purposes. However, to maintain the confidentiality of sensitive data, measures are implemented to encrypt PHI/PII before the logs are stored. This ensures that while users can access these logs for legitimate purposes, they are unable to view or retrieve any sensitive personal information contained within the data payload.



Use Case 2: The procedure entails transferring data from a server within a trusted domain to an external server outside of this domain, using an HTTPS connection to ensure secure data transit. Similar to the internal process, all HTTP activity is meticulously logged, capturing the details of the data exchange without compromising security. Although the destination server lies outside the trusted domain, the integrity of the data in transit is maintained through HTTPS encryption. As with internal logging, any sensitive information within the payload, such as PHI/PII, is encrypted before logging to ensure it remains invisible and inaccessible to users who have permission to view the HTTP logs. This practice upholds privacy and security standards even when interacting with servers beyond the trusted domain.

Use Case 3: In this scenario, data is being transmitted from a server within a trusted domain to another server situated outside the trusted domain. The data transfer may pass through an unspecified number of intermediary systems. Throughout this transmission chain, each segment of the data's journey is secured using HTTPS connections to ensure encryption and integrity of the data in transit. As is standard practice, HTTP activities—including those across the intermediary systems—are comprehensively logged. These logs detail the data exchange process without compromising the encrypted state of sensitive content. Precautions are taken to obscure or strip any Personal Health Information (PHI) or Personally Identifiable Information (PII) from these logs, safeguarding the sensitive data from exposure to users who have access to these logs, thus maintaining privacy and compliance with data protection regulations.

3.2. Data:



SV.Member	
PK	MemberID [Unique Identifier] NOT NULL
	FirstName [nvarchar] NULL
	LastName [nvarchar] NULL
	SSN [nvarchar] NULL Encrypted
	DOB [Date] NULL
	MiddleInitial [nvarchar] NULL
	Gender[nvarchar] NULL
	Email [nvarchar] NULL
	CellPhone[nvarchar] NULL
	Session Key [varchar] NULL

The database component of the Secure Module/Key Management Application is designed to handle the secure storage and retrieval of encrypted data efficiently. Utilizing Microsoft SQL Server, the database architecture is meticulously crafted to support the high-security requirements of the system, ensuring that sensitive information such as Personal Health Information (PHI) and Personally Identifiable Information (PII) is stored in an encrypted format. Rigorous access controls and encryption protocols are applied to protect data integrity and confidentiality. The database is also configured for scalability and performance, with provisions for future enhancements like advanced data masking and real-time data analytics. This robust setup not only provides a secure repository for sensitive data but also ensures that the system can handle growing data volumes as the application scales.

The database schema is a table named SV.Member. Here's a breakdown of its structure and each field:

1. **MemberID [Unique Identifier] NOT NULL**
   a. This is the primary key of the table, which uniquely identifies each record in the table. It cannot be NULL, meaning every record must have a MemberID.
2. **FirstName [nvarchar] NULL**
   a. Stores the first name of the member. It can be NULL, meaning this field is optional.

3. **LastName [nvarchar] NULL**
    a. Stores the last name of the member. This field is also optional.
4. **SSN [nvarchar] NULL Encrypted**
    a. Stores the Social Security Number of the member. It is optional and encrypted for security.
5. **DOB [Date] NULL**
    a. Stores the date of birth of the member. This field is optional.
6. **MiddleInitial [nvarchar] NULL**
    a. Stores the middle initial of the member. This field is optional.
7. **Gender [nvarchar] NULL**
    a. Stores the gender of the member. This field is optional.
8. **Email [nvarchar] NULL**
    a. Stores the email address of the member. This field is optional.
9. **CellPhone [nvarchar] NULL**
    a. Stores the cell phone number of the member. This field is optional.
10. **Session Key [varchar] NULL**
    a. Stores a session key associated with the member. This field is optional.

3.3. Implementation:

The Secure Module/Key Management Application project was divided into four major sections to streamline development and enhance focus on critical areas: Key Management, Data Encryption and Transmission, User Interface (UI), and Integration with Azure Key Vault.

3.3.1. Key Management

The project leverages Microsoft Azure Key Vault to manage cryptographic keys and secrets used for data encryption. Azure Key Vault provides a secure location to store and manage these keys away from the user's direct access, which enhances the security of data operations across all use cases. Key rotation policies were implemented to ensure keys were changed periodically, increasing security.

3.3.2. Data Encryption and Transmission

Data encryption is performed using Advanced Encryption Standard (AES) and Rivest–Shamir–Adleman (RSA) algorithms. AES is used for encrypting the data itself, while RSA is utilized for the encryption of AES keys. Data transmission between servers, whether within a trusted domain or through intermediary systems, is secured with HTTPS to prevent unauthorized access and ensure integrity.

3.3.3. User Interface/User Experience

A specialized user interface was designed to facilitate secure data input and management without exposing underlying security mechanisms to the end-user. This interface includes forms for data entry and views for monitoring data transfer statuses, incorporating robust error handling and user feedback mechanisms to enhance usability and user satisfaction.

3.3.4. Integration with Azure Key Vault

The integration with Azure Key Vault is central to the project, involving configurations to authenticate, store, and manage cryptographic keys and certificates securely. The system's backend interacts with Azure Key Vault via API calls, which handle the retrieval and updating of keys as needed for encryption and decryption processes. This setup ensures that all cryptographic materials are handled according to best security practices.

4. Conclusions:

4.1. Results:

The application successfully met its core objectives of securely managing and transferring sensitive data across different domains. The integration of Microsoft Azure Key Vault effectively managed cryptographic keys, enhancing overall security. The encryption mechanisms employing AES and RSA algorithms functioned as expected, securing data in transit and at rest. However, the project faced challenges with interface intuitiveness and compliance with stringent regulations such as HIPAA, highlighting areas for improvement. User feedback indicated satisfaction with the security and functionality but suggested enhancements for user experience and accessibility. These results will guide future development priorities, focusing on refining UI/UX and expanding compliance features to meet broader industry standards.

1. ASP.NET Development
    - We successfully build a web application using the ASP.NET framework on the .NET platform, utilizing C# as the primary programming language
    - This development approach provided us with a stable structure for building scalable and dynamic web applications
2. HTML Helpers
    - We implemented HTML helpers to facilitate the generation of HTML content within views, making the process more straightforward
    - These helpers significantly contributed to maintaining clean, readable, and concise code, improving the manageability and maintainability of the application's views
3. Razor Syntax (cshtml)
    - By incorporating Razor syntax, we enabled the embedding of server-based code into webpages, allowing for dynamic HTML content generation
    - This approach streamlined the integration of server code with HTML, enhancing the application's ability to generate and manage dynamic web content effectively

4.2. Future:

Compliance and Standards Adaptation:

- Global Compliance Features: Enhancements to support broader compliance standards such as GDPR for European users, in addition to HIPAA.
- Regular Security Audits: Implementing a schedule for regular security audits and compliance checks to adapt to new regulations as they emerge.

Performance Optimization:

- Database Performance Tuning: Optimization of database interactions to handle larger data volumes and reduce latency in data retrieval and encryption operations.
- Load Balancing: Implementation of load balancing techniques to distribute data processing loads evenly across servers, improving response times and system reliability.

User Interface and Accessibility Improvements:

- Adaptive UI: Development of an adaptive user interface that adjusts to various user needs and accessibility standards, enhancing the user experience for individuals with disabilities.
- Mobile Compatibility: Extending the application's interface to be fully functional on mobile devices, accommodating secure management tasks on the go.

5. References:

Get started with Key Vault certificates | Microsoft Learn
https://learn.microsoft.com/en-us/azure/key-vault/certificates/certificate-scenarios

Certificate creation methods | Microsoft Learn
https://learn.microsoft.com/en-us/azure/key-vault/certificates/create-certificate

Using HashiCorp Vault C# client with .NET Core
https://developer.hashicorp.com/vault/tutorials/app-integration/dotnet-httpclient

Azure encryption overview | Microsoft Learn
https://learn.microsoft.com/en-us/azure/security/fundamentals/encryption-overview

Encrypt and Decrypt Data with C# and SQL Server [closed]
https://stackoverflow.com/questions/7921943/encrypt-and-decrypt-data-with-c-sharp-a
nd-sql-server