Senior Design Final Report

Secure Module/Key Management Application

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:

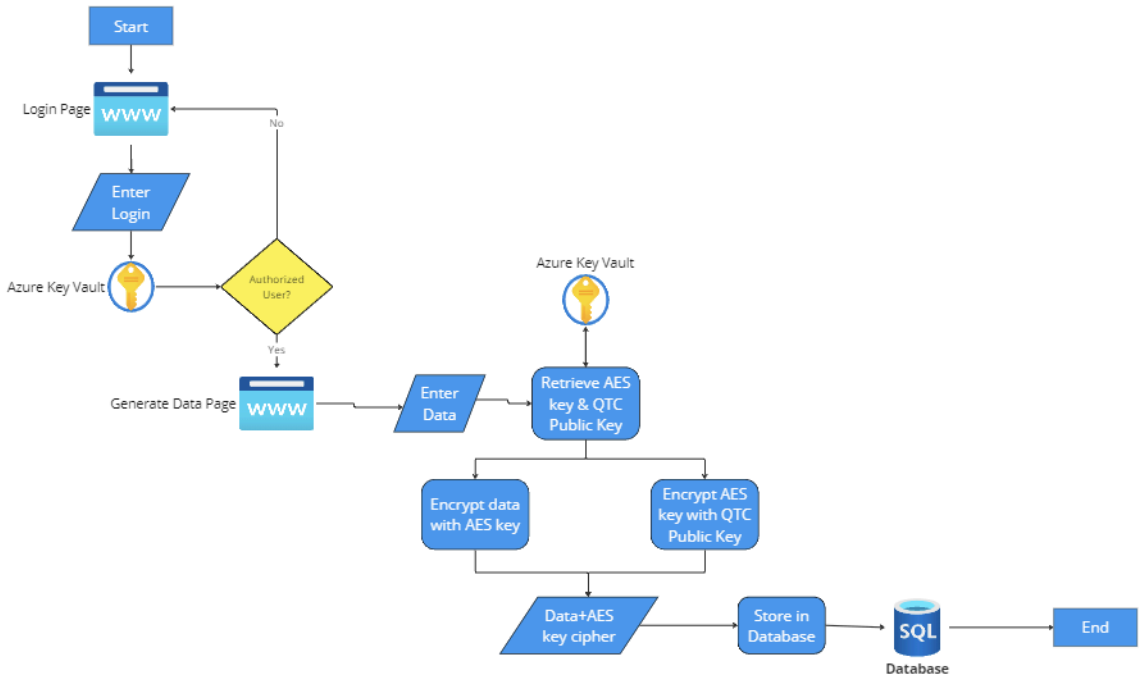
- Ongoing optimization of the system's performance under various loads is crucial, particularly focusing on reducing latency and increasing throughput for high-volume data transfers.

3. System architecture:

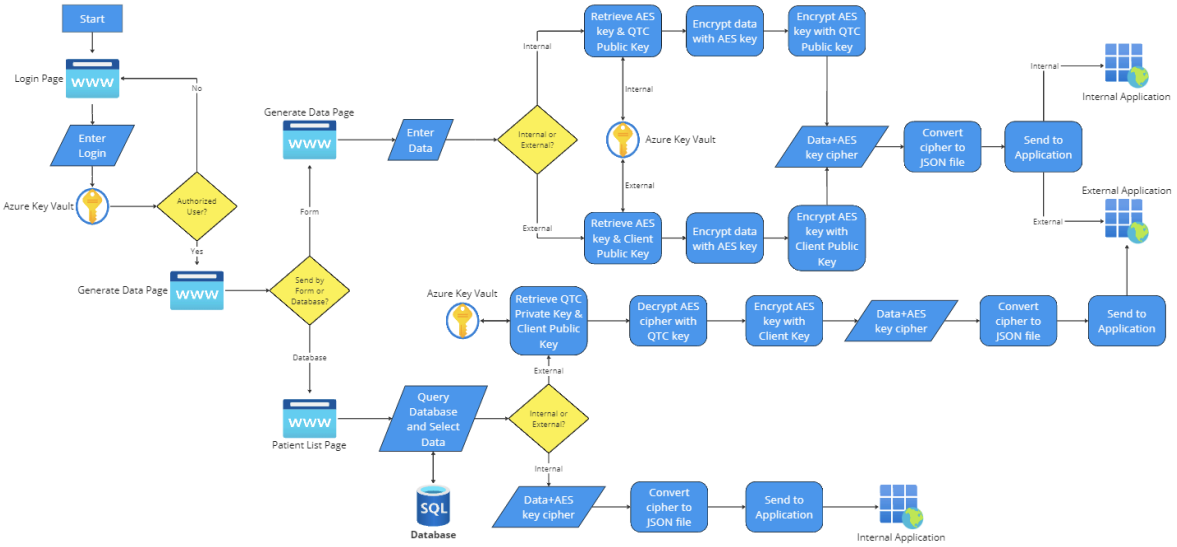
3.1. Overview:

The architecture for the web application.

Here is a diagram that shows how this architecture works:



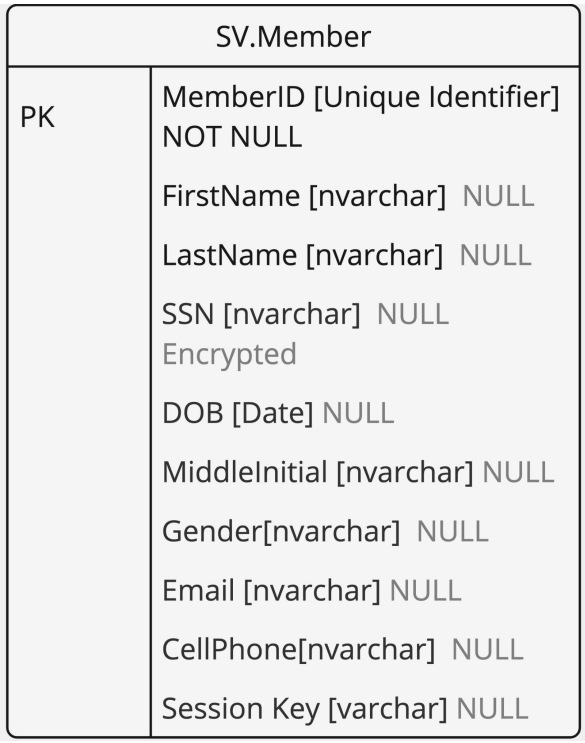
Use Case 1: The process involves the secure transmission of data containing sensitive information (such as Personal Health Information [PHI] and Personally Identifiable Information [PII]) between servers within a trusted domain. The communication channel is safeguarded using HTTPS to ensure encryption 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:



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-and-sql-server