Application Security (Written Assignment)

Full Name: Arshi Lamba
Admin No: 232619X
Tutorial Group: IT2163-02

I have chosen the following security features for my assignment:

1. Payment page with Credit Card info (Pg 2-9)
2. Data encryption archival

Statement on Plagiarism

I certify that this assignment/report is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this assignment/report has not previously been submitted for assessment in any other unit, and that I have not copied in part or whole or otherwise plagiarised the work of other students and/or persons.

Signature	
Date	

Payment Page with Credit Card Info

- *Security Features*
  1) **Tokenization**
     Payment tokenization is a process that replaces sensitive data, like a credit card number, with a non-sensitive, randomly generated token.

     How is tokenization implemented in payment gateways?

     1. Customers would initiate a transaction through an online checkout page. They would enter card details into the payment gateway on the website.

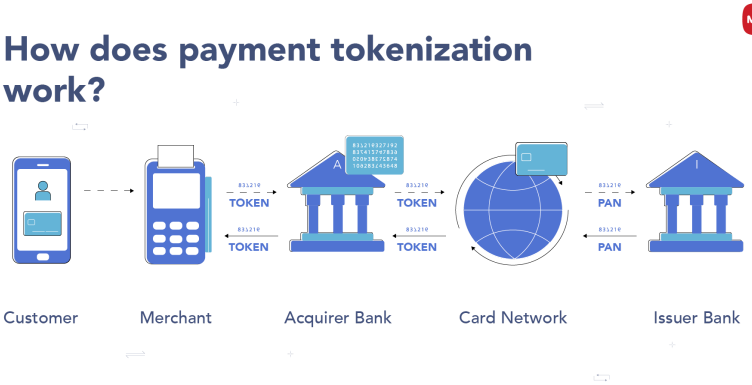
     2. The merchant's payment gateway then sends a request to the payment service provider, which tokenizes the customer's credit card information.

     3. The PCI-compliant payment service provider returns the token reference to the merchant and stores the token mapping to the payment credential data.

     4. The merchant's payment gateway may provision a network token for the card and use the network token - instead of the original card data - to request payment authorization from the card scheme and the customer's bank.

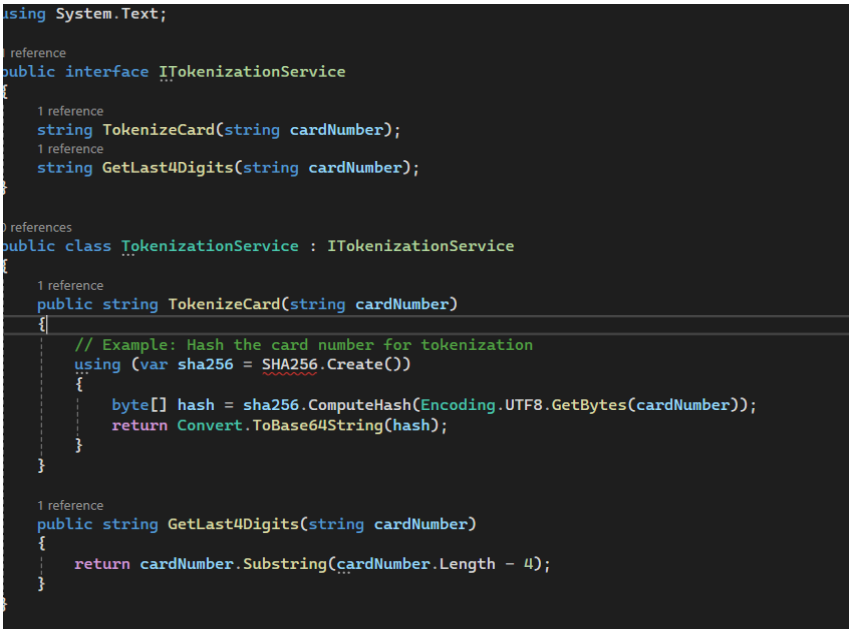
     5. After the issuing bank successfully authorizes the payment, they notify the merchant, and the payment is completed.

     6. The merchant can then store the token for future transactions from that customer - be it for recurring payments, refunds, or to enable one-click payments - without failing foul of PCI DSS (Payment Card Industry Data Security Standard) compliance requirements.



Sample Code:

This shows the service to tokenize Credit Card Details in the database.



```
using System.Text;

reference
public interface ITokenizationService
{
    1 reference
    string TokenizeCard(string cardNumber);
    1 reference
    string GetLast4Digits(string cardNumber);
}

0 references
public class TokenizationService : ITokenizationService
{
    1 reference
    public string TokenizeCard(string cardNumber)
    {
        // Example: Hash the card number for tokenization
        using (var sha256 = SHA256.Create())
        {
            byte[] hash = sha256.ComputeHash(Encoding.UTF8.GetBytes(cardNumber));
            return Convert.ToBase64String(hash);
        }
    }

    1 reference
    public string GetLast4Digits(string cardNumber)
    {
        return cardNumber.Substring(cardNumber.Length - 4);
    }
}
```

Security issue(s) or problems that are avoided or mitigated using tokenization

1. Data Breaches

   As of 2024, the average data breach cost in the United States amounted to $9.36 million, therefore, protecting sensitive payment data is critical. Payment tokenization renders primary account numbers (PANs) and other sensitive payment details useless for fraudulent activities by replacing them with secure, non-sensitive tokens during transactions. Fraudsters often target businesses that store payment card data or process payments without sufficient security measures, such as encryption or tokenization.

2. Compliance Requirements

   Payment tokenization significantly reduces merchants' exposure to sensitive payment data. It eliminates the need to store original cardholder data, keeping transactions secure and simplifying compliance with PCI DSS and other data protection regulations, such as GDPR and CCPA.

3. Contactless Payments and IoT Transactions

As contactless payments and IoT-based transactions grow in adoption, they introduce new attack vectors for cybercriminals. By isolating sensitive data from the tokenization process, payment tokenization significantly reduces the attack surface while ensuring convenience and scalability for emerging technologies.

Why is this solution effective?

Tokenization reduces risk from data breaches, helps foster customer trust, minimizes red tape, and drives technology behind popular payment services, like mobile wallets.

2) Content Security Policy (CSP)

CSP is a widely supported Web security standard intended to prevent certain types of injection-based attacks by giving developers control over the resources loaded by their applications.

How is Content Security Policy (CSP) implemented in the payment gateway?

Sample Code:

```
using Microsoft.AspNetCore.Mvc;
using Org.BouncyCastle.Asn1.Ocsp;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Web.Http;

0 references
public class PaymentController : ApiController
{
    [HttpGet]
    [Route("api/payment")]
    0 references
    public HttpResponseMessage GetPaymentPage()
    {
        var response = Request.CreateResponse(HttpStatusCode.OK, "Payment Page Content");

        response.Headers.Add("Content-Security-Policy",
            "default-src 'none'; " +
            "script-src 'self' https://api.example-payment-processor.com; " +
            "style-src 'self'; " +
            "img-src 'self'; " +
            "font-src 'self'; " +
            "frame-ancestors 'none'; " +
            "connect-src https://api.example-payment-processor.com;");

        return response;
    }
}
```

Explanation:

1. Import necessary namepaces:
    - System.Net: Provides classes for network communication

- System.Net.Http: Provides classes for making HTTP requests and responses
- System.Net.Http.Headers: Provides classes for working with HTTP headers
- System.Web.Http: Provides classes for building and handling HTTP requests and responses in ASP.NET Web API

2. Create a Controller
   - Create a controller class, in this case PaymentController, that will handle the payment page request.
3. Define the Route:
   - Use the [HttpGet] and [Route("api/payment")] attributes to define the HTTP GET method and the route for the payment page.
4. Create a HTTP response
   - Create an HttpResponseMessage object to represent the HTTP response.
   - Set the content of the response to the HTML content of the payment page.
5. Add CSP Header
   - Use the response.Headers.Add() method to add the Content-Security-Policy header to the response.
   - Return the HttpResponseMessage object to the client.

Security issue(s) or problems that are avoided or mitigated using CSP

1. Cross-Site Scripting (XSS)
   Attackers inject malicious scripts into the payment gateway webpages. These scripts can steal sensitive data like credit card, CVV codes, and other personal information when executed by the user's browser. CSP prevents the execution from untrusted sources. This effectively blocks XSS attacks as malicious scripts cannot be loaded and run.

   Example Code: This prevents the loading of images from external sources, minimizing the risk of injected styles affecting the payment page's layout or behavior.

<meta http-equiv="Content-Security-Policy" content="style-src 'self'">

2. Data Injection

Attackers can inject malicious content into the payment page through vulnerabilities. This injected content can manipulate user input or steal sensitive data. Only allowed resources from trusted sources can be loaded, reducing the risk of data injection attacks.

Why is this solution effective?

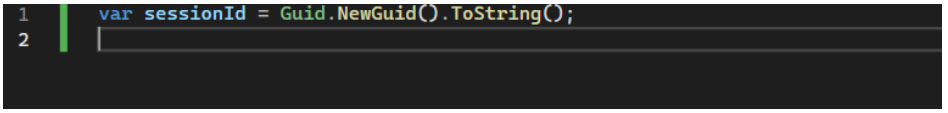
CSP is a crucial security mechanism for payment gateways due to its ability to effectively mitigate common web-based attacks, protect sensitive user data, and enhance the overall security posture. By carefully defining and enforcing a robust CSP, payment gateway providers can significantly reduce their risk exposure and provide a more secure and trustworthy experience for their customers.

3) **Session Management**

Session management in a payment gateway allows a user to provide their payment details directly to the gateway.

How is Session Management implemented in the payment gateway?

1. Create a Unique Session ID
   - Generate a unique identifier for the session upon initiating a transaction
   - Example in .NET:



2. Store the Session ID Securely
   - Store the session ID on the client side (browser cookies) and server-side for tracking
3. Implement Secure Session Handling
   - Enforce HTTPS for all communication to encrypt session data during transmission.
   - Set Secure, which ensure that cookies are only sent over HTTPS and HttpOnly Cookies Attributes, which prevents JavaScript from accessing cookies.
   - Example in ASP .NET Core:

```
using MySqlX.XDevAPI;

var cookieOptions = new CookieOptions
{
    Secure = true,
    HttpOnly = true,
    SameSite = SameSiteMode.Strict
};
Response.Cookies.Append("SessionID", sessionId, cookieOptions);
```

4. Store Session Data Securely
    - Use a server-side store to store session data in memory, a database, or a distributed cache.
    - Session Store Example:

```
HttpContext.Session.SetString("TransactionID", transactionId.ToString());
```

5. Regenerate Session on Critical Events
    - To prevent Session Fixation, regenerate the session ID upon user authentication or important events
    - Example:

```
HttpContext.Session.Clear(); // Clear existing session
var newSessionId = Guid.NewGuid().ToString(); // Generate new session ID
```

6. Use Session Tokens for Sensitive Data
    - Represent session data with tokens to avoid exposing sensitive details.
    - Example: Use JWT or cryptographic tokens
7. Monitor and Secure Sessions
    - Track session activities for anomalies
    - Use Multi-Factor Authentication (MFA) for high-risk transactions to enhance security.
8. Test for vulnerabilities
    - Perform security tests for session-related vulnerabilities like fixation, hijacking, and cross-site scripting (XSS)
9. Integrate Session in Payment Gateway
    - Use hosted sessions where the payment gateway securely handles sensitive data
    - Example: Mastercard or Stripe's hosted payment page
    - Use payment gateway APIs to create and manage sessions

- Example:

```
var sessionResponse = await paymentGatewayClient.CreateSessionAsync(request);
```

10. Handle Session Termination
    - Clear session data after a transaction is completed or expired
    - Example:

```
HttpContext.Session.Clear();
```

Security issue(s) or problems that are avoided or mitigated using Session Management

1. Session Hijacking
   Hijacking occurs when an attacker compromises a session token by stealing or predicting it to gain unauthorized access to the server on behalf of its owner. There are a few ways in which an attacker can achieve this. One method is using XSS. If the web application the user is accessing has the httpOnly attribute set to false, the level of exposure to session hijacking through the client side is very high, creating an opening for XSS attacks to occur.

2. Session Fixation
   Fixation occurs when an attacker tricks the user into using an already existing session ID to access the server by hijacking the valid user session. It mainly occurs when the user accessing the session is not properly validated using the session ID. It can also occur when an attacker sends the victim a vulnerable session ID within a URL.

Why is this solution effective?

Effective session management is crucial for maintaining the security, performance, and scalability of web applications. As users navigate through web pages, their interaction needs to be seamlessly

connected and maintained.

Data Encryption and Archival

- *Security Features*
    1) Advanced Encryption Standards (AES)

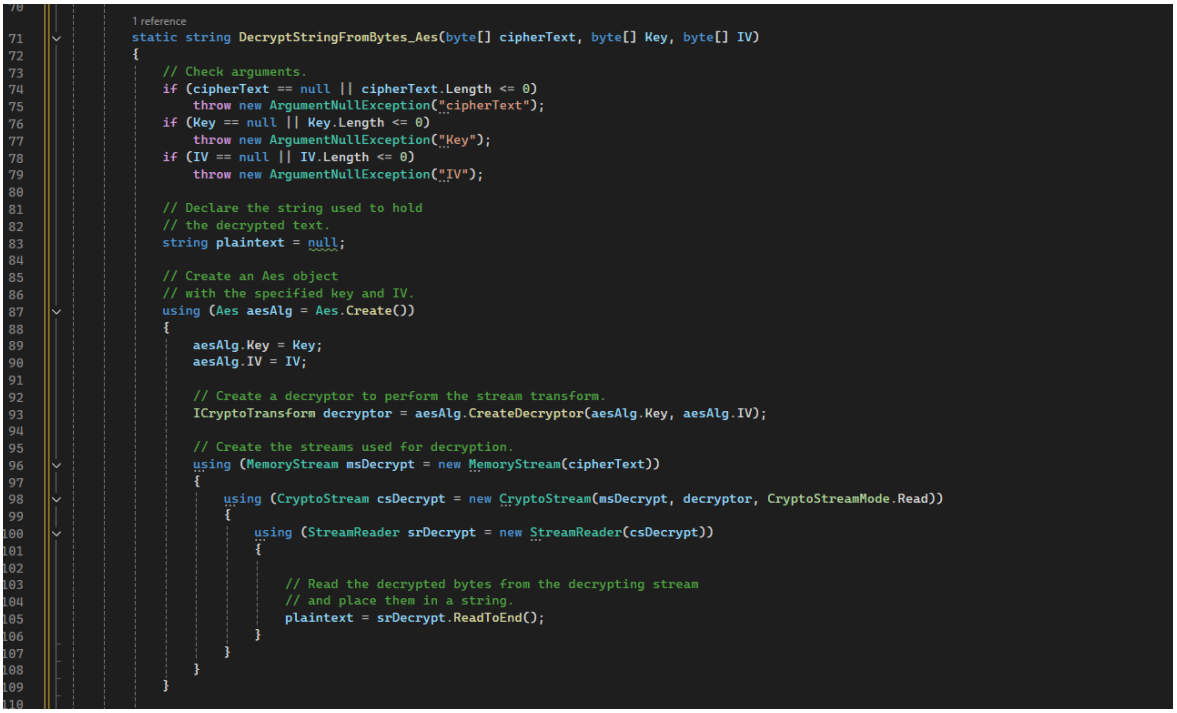
AES is implemented in software and hardware throughout the world to encrypt sensitive data. It is essential for government computer security, cybersecurity, and electronic data protection. Since AES puts data through multiple encryption rounds and splits a message into smaller blocks of 128 bits, it is more secure and reliable than older symmetric encryption methods.

How is AES implemented in Data Encryption and Archival?

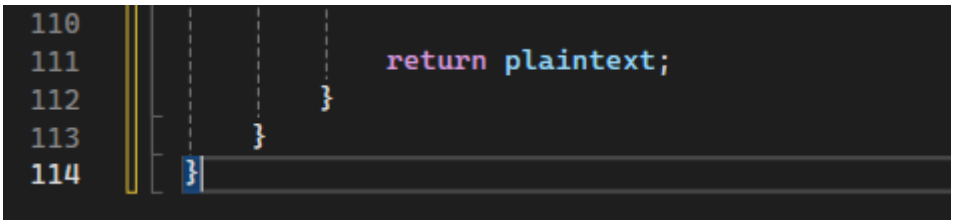
1. Choose AES Parameters
    - Key Size: AES supports three key sizes:
      > 128-bit: Fastest but offers less security
      -> 192-bit: Offers a balance between performance and security
      -> 256-bit: Highly secure and recommended for sensitive data
    - Mode of Operation
      -> CBC (Cipher Block Claiming): Popular mode that uses an Initialization Vector (IV) for randomized encryption
      -> CTR (Counter Mode): Suitable for parallel processing
2. Generate AES Keys
    - Generate keys using a Cryptographically Secure RNG (CSPRNG) to ensure unpredictability
    - Store Keys securely in:
      -> Hardware Security Modules (HSMs): Specialized hardware for secure key storage
      -> Key Management Systems (KMS): Tools like AWS KMS or Azure Key Vault for managing the key lifecycle
    - Never hard-code keys in the application or store them in plaintext
3. Encrypt Data
    - Randomly generate an IV for each encryption session
    - Ensure the IV is unique but not secret
    - Split plaintext generated into 128-bit blocks
    - Apply the AES algorithms with the chosen mode and padding scheme
    - Example Code: (shows the entire process of AES)

```
using System;
using System.IO;
using System.Security.Cryptography;

namespace Aes_Example
{
    0 references
    class AesExample
    {
        0 references
        public static void Main()
        {
            string original = "Here is some data to encrypt!";

            // Create a new instance of the Aes
            // class.  This generates a new key and initialization
            // vector (IV).
            using (Aes myAes = Aes.Create())
            {

                // Encrypt the string to an array of bytes.
                byte[] encrypted = EncryptStringToBytes_Aes(original, myAes.Key, myAes.IV);

                // Decrypt the bytes to a string.
                string roundtrip = DecryptStringFromBytes_Aes(encrypted, myAes.Key, myAes.IV);

                //Display the original data and the decrypted data.
                Console.WriteLine("Original:   {0}", original);
                Console.WriteLine("Round Trip: {0}", roundtrip);
            }
        }
```

```
        1 reference
        static byte[] EncryptStringToBytes_Aes(string plainText, byte[] Key, byte[] IV)
        {
            // Check arguments.
            if (plainText == null || plainText.Length <= 0)
                throw new ArgumentNullException("plainText");
            if (Key == null || Key.Length <= 0)
                throw new ArgumentNullException("Key");
            if (IV == null || IV.Length <= 0)
                throw new ArgumentNullException("IV");
            byte[] encrypted;

            // Create an Aes object
            // with the specified key and IV.
            using (Aes aesAlg = Aes.Create())
            {
                aesAlg.Key = Key;
                aesAlg.IV = IV;

                // Create an encryptor to perform the stream transform.
                ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);

                // Create the streams used for encryption.
                using (MemoryStream msEncrypt = new MemoryStream())
                {
                    using (CryptoStream csEncrypt = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write))
                    {
                        using (StreamWriter swEncrypt = new StreamWriter(csEncrypt))
                        {
                            //Write all data to the stream.
                            swEncrypt.Write(plainText);
                        }
                    }

                    encrypted = msEncrypt.ToArray();
                }
            }

            // Return the encrypted bytes from the memory stream.
            return encrypted;
        }
```



```
1 reference
static string DecryptStringFromBytes_Aes(byte[] cipherText, byte[] Key, byte[] IV)
{
    // Check arguments.
    if (cipherText == null || cipherText.Length <= 0)
        throw new ArgumentNullException("cipherText");
    if (Key == null || Key.Length <= 0)
        throw new ArgumentNullException("Key");
    if (IV == null || IV.Length <= 0)
        throw new ArgumentNullException("IV");

    // Declare the string used to hold
    // the decrypted text.
    string plaintext = null;

    // Create an Aes object
    // with the specified key and IV.
    using (Aes aesAlg = Aes.Create())
    {
        aesAlg.Key = Key;
        aesAlg.IV = IV;

        // Create a decryptor to perform the stream transform.
        ICryptoTransform decryptor = aesAlg.CreateDecryptor(aesAlg.Key, aesAlg.IV);

        // Create the streams used for decryption.
        using (MemoryStream msDecrypt = new MemoryStream(cipherText))
        {
            using (CryptoStream csDecrypt = new CryptoStream(msDecrypt, decryptor, CryptoStreamMode.Read))
            {
                using (StreamReader srDecrypt = new StreamReader(csDecrypt))
                {

                    // Read the decrypted bytes from the decrypting stream
                    // and place them in a string.
                    plaintext = srDecrypt.ReadToEnd();
                }
            }
        }
    }
```



```
        return plaintext;
    }
}
}
```

4. Store Encrypted Data
    - Store encrypted data in a secure archival system such as database (e.g. PostgreSQL, MySQL) and file systems (e.g. encrypted storage or secure S3 buckets)
    - Store associated metadata with the ciphertext
    - Use redundant storage for disaster recovery
5. Decrypt Data
    - Retrieve the AES key, IV and metadata stored with the ciphertext
    - Decrypt the data using the same AES parameters used for encryption
6. Ensure Data Integrity
    - Use AES-GCM which includes an authentication tag to verify data integrity during decryption
    - Ensure that tampered ciphertext fails decryption
7. Key Rotation and Management
    - Periodically rotate keys to minimize the impact of key compromise

- Use KMS to automate key rotation and enforce strong key policies

8. Compliance and Security
   - Follow standards like:
     -> GDPR: Ensures personal data encryption for privacy
     -> HIPAA: Mandates encryption for healthcare data
     -> PCI DSS: Requires encryption for credit card data
   - Log all encryption and decryption activities for audits

Security issue(s) or problems that are avoided or mitigated using AES

1. Data Confidentiality
   AES uses symmetric encryption to convert plaintext into ciphertext making data unintelligible to unauthorized parties without the encryption key. Strong key lengths ensure that brute-force attacks are computationally infeasible.

2. Eavesdropping and Interception
   Encrypted data over communication channels ensures that even if intercepted, the data remains unreadable without the decryption key.

3. Side-Channel Attacks
   AES is designed to be efficient for both hardware and software implementations, and secure implementations reduce vulnerability to side-channel attacks.

Why is this solution effective?

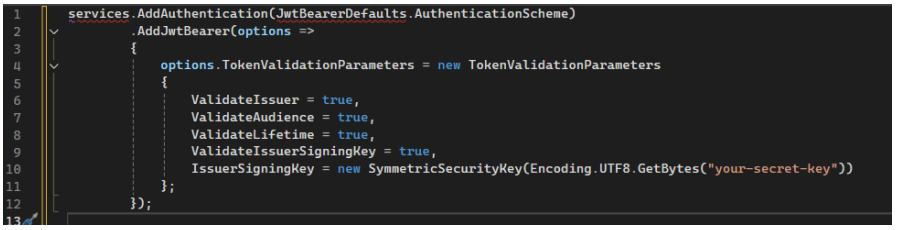
AES is a highly effective encryption standard due to its strong security design, including resistance to cryptographic attacks and support for 128, 192, and 256-bit keys. It is efficient, scalable, and widely adopted in protocols like SSL/TLS and WPA3, ensuring data confidentiality and integrity. AES supports versatile modes of operation, such as CBC and GCM, enabling use across various encryption needs. Its rigorous testing by NIST and global acceptance make it a trusted choice for sensitive data protection.

2) Access Controls and Authentication
   Authentication is any process by which a system verifies the identity of a user who wishes to access the system. Because access control is typically based on the identity of the user who requests access to a resource, authentication is essential to effective security.

Access control works by identifying and regulating the policies for accessing particular resources and the exact activities that users can perform within those resources. This is done by the process of authentication, which is the process of establishing the identity of the user, and the process of authorization, which is the process of determining what the authorized user is capable of doing. It can occur at various levels, such as network level, application level, and physical level, in relation to buildings and other structures.

How are Access Controls and Authentication implemented in Data Encryption and Archival?

1. User Authentication
    - Implement robust authentication mechanisms such as multi-factor authentication to verify the identity of users before allowing access to sensitive data.
    - Example Code:



```
services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
        .AddJwtBearer(options =>
        {
            options.TokenValidationParameters = new TokenValidationParameters
            {
                ValidateIssuer = true,
                ValidateAudience = true,
                ValidateLifetime = true,
                ValidateIssuerSigningKey = true,
                IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("your-secret-key"))
            };
        });
```

2. Role-Based Access Control
    - Assign users specific roles and define access privileges based on those roles to determine who can encrypt, decrypt and access archived data.
    - Example Code:

```
services.AddAuthorization(options =>
{
    options.AddPolicy("RequireAdminRole", policy => policy.RequireRole("Admin"));
});
```

3. Encryption Key Management
    - Ensure encryption keys are securely stored and access to these keys is restricted.
4. Auditing and Logging
    - Implement logging mechanisms to track all access and modification attempts to encrypted data, allowing for detection of unauthorized access
5. Data Access Policies
    - Define clear data access policies, ensuring that only authorized individuals or systems can access sensitive data.

Security issue(s) or problems that are avoided or mitigated using AES

3) Key Management

References:

https://stripe.com/en-sg/resources/more/payment-security
https://newsroom.mastercard.com/news/perspectives/2024/what-is-tokenization/
https://www.aciworldwide.com/blog/a-primer-on-tokens-tokenization-payment-tokens-and-merchant-tokens
https://www.checkout.com/blog/payment-tokenization
https://www.memcyco.com/what-is-payment-tokenization-for-information-security/#:~:text=Payment%20tokenization%20significantly%20reduces%20merchants,such%20as%20GDPR%20and%20CCPA.
https://www.worldpay.com/en/insights/articles/4-reasons-to-use-tokenization
https://developers.google.com/tag-platform/security/guides/csp
https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html
https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP
https://www.w3.org/TR/CSP3/
https://verpex.com/blog/privacy-security/content-security-policies-csp-implementation-and-benefits#:~:text=Preventing%20Code%20Injection%3A%20CSP%20helps,defense%20against%20new%20attack%20vectors.
https://ap-gateway.mastercard.com/api/documentation/integrationGuidelines/hostedSession/paymentSessions.html?locale=en_US#:~:text=You%20can%20use%20a%20session%20to%20allow,payment%20details%20directly%20to%20the%20Mastercard%20Gateway.
https://www.google.com/url?sa=E&source=gmail&q=https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html
https://learn.microsoft.com/en-us/aspnet/core/fundamentals/app-state?view=aspnetcore-7.0
https://docs.stripe.com/api/checkout/sessions
https://snyk.io/blog/session-management-security
https://www.descope.com/learn/post/session-management
https://www.techtarget.com/searchsecurity/definition/Advanced-Encryption-Standard
https://csrc.nist.gov/publications/detail/fips/197/final
https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation
https://owasp.org/www-project-cheat-sheets/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html
https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation
https://learn.microsoft.com/en-us/dotnet/api/system.security.cryptography.aes
https://en.wikipedia.org/wiki/Galois/Counter_Mode
https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf
https://en.wikipedia.org/wiki/Advanced_Encryption_Standard
https://www.ibm.com/docs/en/wca/3.0.0?topic=security-authentication-versus-access-control

https://www.sentinelone.com/cybersecurity-101/cybersecurity/what-is-access-control/#:~:text=DSS%2C%20among%20others.-,How%20Access%20Control%20Works?,to%20buildings%20and%20other%20structures.
https://learn.microsoft.com/en-us/aspnet/core/security/authentication/
https://aws.amazon.com/kms/
https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html