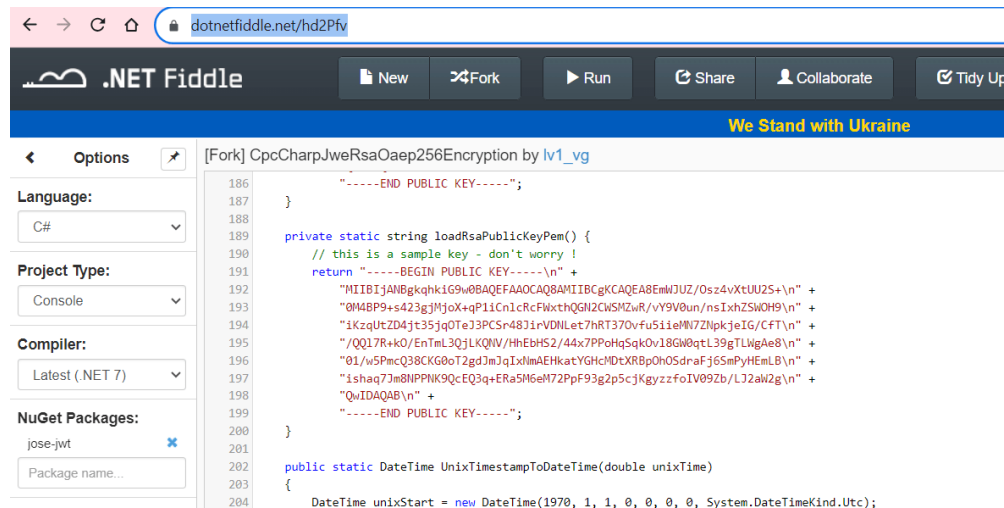


Top 2: Hands On

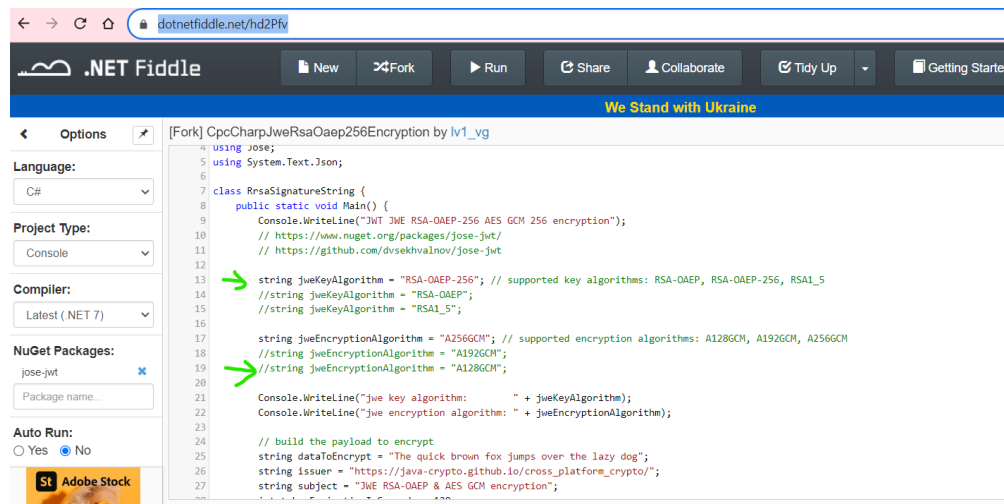
For testing Cryptographic Weakness you have to manually review the code.

Point 1: Browse Dotnet Fiddle site, <https://dotnetfiddle.net/hd2Pfv> and check the crypto key length (strong length minimum 256 bytes /2048 bits), like public key or private key length should not short.



```
186         "-----END PUBLIC KEY-----";
187     }
188
189     private static string loadRsaPublicKeyPem() {
190         // this is a sample key - don't worry !
191         return "-----BEGIN PUBLIC KEY-----\n" +
192             "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA8EmWJUZ/Osz4vXtUu2S+\n" +
193             "0M4BP9+s423gjHjoX+qP1iCn1cRcFwXthQGN2CW5M2wR/vY9V0un/nsIxhZSMOH9\n" +
194             "iKzqUtZD4jt35jq0Te13PC5r483jrVDNLt7hRT370vfu51eMN7ZlNpkjeIG/CFT\n" +
195             "/QQ17R+k0/EnTmL3QjLKQNV/HhEhS2/44x7PPoHq5gkOv18QW0qtL39gTLHgAe8\n" +
196             "01/w5PmcQ38CKG0oT2gdJm3qIXhMAEHkatYGHcMDtXRBp0hO5draFj6SmPyHEmLB\n" +
197             "isshaq7Jm8NPPNK9QcEQ3q+ERa5M6eM72PpF93g2p5cjKgyzzfoIV092b/LJ2aW2g\n" +
198             "QuIDAQAAB\n" +
199             "-----END PUBLIC KEY-----";
200     }
201
202     public static DateTime UnixTimestampToDateTime(double unixTime)
203     {
204         DateTime unixStart = new DateTime(1970, 1, 1, 0, 0, 0, 0, System.DateTimeKind.Utc);
```

Point 2: Verify Strong Cryptography



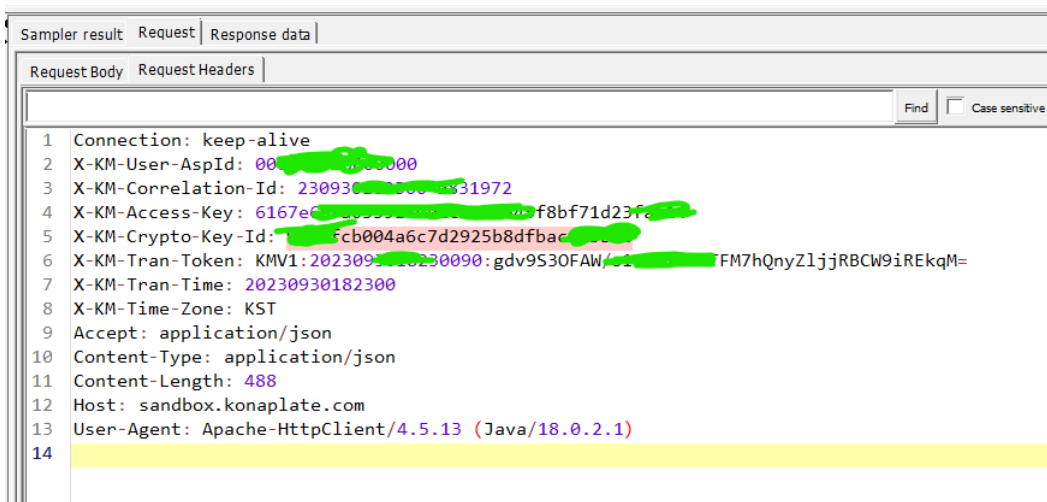
```
4 using Jose;
5 using System.Text.Json;
6
7 class RsaSignatureString {
8     public static void Main() {
9         Console.WriteLine("JWT JWE RSA-OAEP-256 AES GCM 256 encryption");
10        // https://www.nuget.org/packages/jose-jwt/
11        // https://github.com/dvsekhvalnov/jose-jwt
12
13        string jweKeyAlgorithm = "RSA-OAEP-256"; // supported key algorithms: RSA-OAEP, RSA-OAEP-256, RSA1_5
14        //string jweKeyAlgorithm = "RSA-OAEP";
15        //string jweKeyAlgorithm = "RSA1_5";
16
17        string jweEncryptionAlgorithm = "A256GCM"; // supported encryption algorithms: A128GCM, A192GCM, A256GCM
18        //string jweEncryptionAlgorithm = "A192GCM";
19        //string jweEncryptionAlgorithm = "A128GCM";
20
21        Console.WriteLine("jwe key algorithm: " + jweKeyAlgorithm);
22        Console.WriteLine("jwe encryption algorithm: " + jweEncryptionAlgorithm);
23
24        // build the payload to encrypt
25        string dataToEncrypt = "The quick brown fox jumps over the lazy dog";
26        string issuer = "https://java-crypto.github.io/cross_platform_crypto/";
27        string subject = "JWE RSA-OAEP & AES GCM encryption";
```

List of Weak Cryptography:

1. Data Encryption Standard (DES): DES was once a widely used symmetric encryption algorithm, but it is now considered weak due to its small key size (56 bits). It is susceptible to brute force attacks, and its use is strongly discouraged.
2. Triple Data Encryption Algorithm (3DES): While 3DES is an improvement over DES, it is considered outdated as well. It uses the same DES algorithm three times with different keys but has vulnerabilities compared to modern encryption algorithms like AES.
3. Rivest Cipher (RC4): RC4 was once used for SSL/TLS encryption in web browsers and other applications, but multiple vulnerabilities have been discovered in its implementation. It is now considered insecure, and its use is discouraged.
4. MD5 (Message Digest 5): MD5 is a hash function that has known collision vulnerabilities. It is no longer considered secure for cryptographic purposes like digital signatures.
5. SHA-1 (Secure Hash Algorithm 1): SHA-1 is another hash function that has known collision vulnerabilities. It is being phased out in favor of more secure hash functions like SHA-256.
6. WEP (Wired Equivalent Privacy): WEP was an early security protocol for wireless networks, but it has well-documented weaknesses. It is easily cracked, and its use is strongly discouraged.

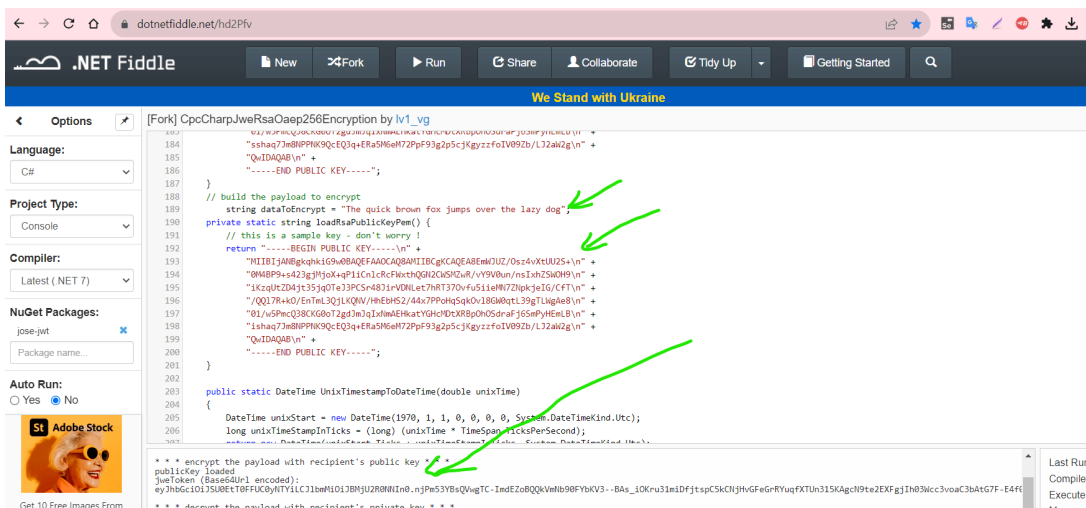
7. SSL and Early TLS Versions: Older versions of the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols, such as SSLv2, SSLv3, and early TLS 1.0, have known vulnerabilities. It is recommended to use modern TLS versions (e.g., TLS 1.2 or TLS 1.3) for secure communication.
8. RSA Key Sizes Below 2048 Bits: As computing power has increased, RSA key sizes below 2048 bits are no longer considered sufficiently secure for long-term protection of data. Key sizes of 2048 bits or higher are recommended.
9. Diffie-Hellman Key Exchange with Small Parameters: Using small parameters in the Diffie-Hellman key exchange can make it vulnerable to attacks. Stronger and larger parameter sizes should be used to enhance security.
10. Insecure Password Hashing Algorithms: Hashing algorithms like MD5 and SHA-1 should not be used for securely hashing passwords. Instead, stronger and slower hash functions like bcrypt or Argon2 should be used.

Point 3: Verify **X-Type** option must in the request headers which should be encrypted



```
1 Connection: keep-alive
2 X-KM-User-AspId: 0000000000000000
3 X-KM-Correlation-Id: 230930182300-31972
4 X-KM-Access-Key: 6167e6f8bf71d23f...
5 X-KM-Crypto-Key-Id: fcb004a6c7d2925b8dfbac...
6 X-KM-Tran-Token: KMV1:2023093018230090:gdv9S30FAW...FM7hQnyZ1jjRBCW9iREkqM=
7 X-KM-Tran-Time: 20230930182300
8 X-KM-Time-Zone: KST
9 Accept: application/json
10 Content-Type: application/json
11 Content-Length: 488
12 Host: sandbox.konaplate.com
13 User-Agent: Apache-HttpClient/4.5.13 (Java/18.0.2.1)
14
```

Point 4: Ensure **Initializing Vector (IV)** is used in that way when different cyphertext will be generated though public key and plaintext is same.



```
[Fork] CpcCharpJweRsaOaep256Encryption by lv1_vg
184 "shaq7Jm8PPHK9QcEQ3q+E8a5M6eH72PpF93g2p5cJKgyzfoIV092b/L32aIdg/n" +
185 "QuIDAQAB/n" +
186 "-----END PUBLIC KEY-----";
187
188 // build the payload to encrypt
189 string dataToEncrypt = "The quick brown fox jumps over the lazy dog";
190
191 private static string loadRsaPublicKeyPem() {
192     // this is a sample key - don't worry !
193     return "-----BEGIN PUBLIC KEY-----\n" +
194         "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA8E6wU7/0s4vX1U2S+Vw" +
195         "QMBP9+s423gJHj0kqP11Cn1cFbctHQ8Q2G592uR/vY9V0un/ns1xh2SHQ9Pln" +
196         "JKzqUc2D4j353q0t3PC5s4831rV0Hlet7HRT370vfu51eN72lpkJeIG/CFTVn" +
197         "/QQ7R+KO/EnTmL3QjLKQW/HHEBMS2/44x7PP0hqSqk0v18Gw0tL39gTLighe8Vn" +
198         "0L/vSPmcQ38CKG8o72gd3mJqIxmE8kATYGHcHdXRBp0h05draFj65mPyHEmLBVn" +
199         "Ishaq7Jm8PPHK9QcEQ3q+E8a5M6eH72PpF93g2p5cJKgyzfoIV092b/L32aIdg/n" +
200         "QuIDAQAB/n" +
201         "-----END PUBLIC KEY-----";
202
203 public static DateTime UnixTimestampToDateTime(double unixTime)
204 {
205     DateTime unixStart = new DateTime(1970, 1, 1, 0, 0, 0, System.DateTimeKind.Utc);
206     long unixTimestampInTicks = (long)(unixTime * TimeSpan.TicksPerSecond);
207     return unixStart.AddTicks(unixTimestampInTicks).ToLocalTime();
208 }
209
210 // encrypt the payload with recipient's public key
211 public string Encrypt(string dataToEncrypt, string publicKey)
212 {
213     RSACryptoServiceProvider rsa = new RSACryptoServiceProvider(2048);
214     rsa.ImportParameters(Convert.FromBase64String(publicKey));
215     byte[] encryptedData = rsa.Encrypt(Encoding.UTF8.GetBytes(dataToEncrypt), false);
216     return Convert.ToBase64String(encryptedData);
217 }
218
219 // decrypt the payload with recipient's private key
220 public string Decrypt(string encryptedData, string privateKey)
221 {
222     RSACryptoServiceProvider rsa = new RSACryptoServiceProvider(2048);
223     rsa.ImportParameters(Convert.FromBase64String(privateKey));
224     byte[] decryptedData = rsa.Decrypt(Convert.FromBase64String(encryptedData), false);
225     return Encoding.UTF8.GetString(decryptedData);
226 }
227
228 // Run the encryption and decryption
229 string publicKeyPem = loadRsaPublicKeyPem();
230 string encryptedData = Encrypt(dataToEncrypt, publicKeyPem);
231 string decryptedData = Decrypt(encryptedData, privateKeyPem);
232 Console.WriteLine("Encrypted Data: " + encryptedData);
233 Console.WriteLine("Decrypted Data: " + decryptedData);
234 }
```

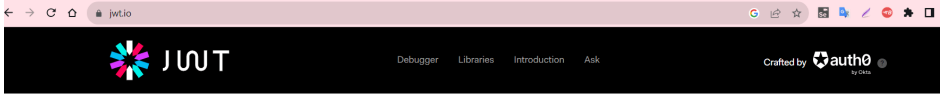
Point 5: Analyzing JWT Session

request e jodhi csrf token thake tahole issue na

Md. Farhan Islam 9/4 5:30 PM

Request e Access Token Ache bhaia

CSRF token name e kichu nai

[illegible]

Encoded PASTE A TOKEN HERE

[illegible]

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS512"
}
```

PAYLOAD: DATA

```

"clientId": "832ac311[REDACTED]cc4d7857",
"created": "1693826617818",
"roleNameValue": "1693826617818",
"roleList": "4264cbac[REDACTED]2e4bc1329a:all-
access",
"type": "ACCESS_TOKEN",
"userId": "81721476[REDACTED]c8db9eaf4b",
"aspid": "10[REDACTED]2345",
"phone": null,
"designation": null,
"userType": "C.SU",
"exp": "1693826917",
"email": null
}

```

Note: A CSRF token can protect CSRF attack though session hijack.