using System;

using System.Collections.Generic;

using System.Text;

using System.Security.Cryptography;

class Program

{

public static void Main()

{

string issuer = "mySystem";

string audience = "myUsers";

string secret = "mysuperdupersecret";

byte[] keyBytes = Encoding.UTF8.GetBytes(secret);

Console.WriteLine("=== JWT DEMO — No JSON Library ===");

string token = GenerateToken(101, "Admin", 2, issuer, audience, keyBytes);

Console.WriteLine("\nGenerated Token:");

Console.WriteLine(token);

bool isValid = ValidateToken(token, keyBytes, issuer, audience, out string payload, out string error);

Console.WriteLine("\nValidation Result:");

Console.WriteLine(isValid ? "VALID ✅" : "INVALID ❌");

Console.WriteLine(isValid ? "Payload: " + payload : "Error: " + error);

string tampered = token.Replace('a', 'b');

Console.WriteLine("\nTampered Token Test:");

Console.WriteLine(ValidateToken(tampered, keyBytes, issuer, audience, out \_, out error) ? "Unexpectedly valid!" : "Tampered — Invalid (" + error + ")");

}

static string GenerateToken(int userId, string role, int minutesValid, string issuer, string audience, byte[] keyBytes)

{

string headerJson = "{\"alg\":\"HS256\",\"typ\":\"JWT\"}";

string payloadJson = $"{{\"sub\":{userId},\"role\":\"{role}\",\"iss\":\"{issuer}\",\"aud\":\"{audience}\",\"exp\":{DateTimeOffset.UtcNow.AddMinutes(minutesValid).ToUnixTimeSeconds()}}}";

string headerB64 = Base64UrlEncode(Encoding.UTF8.GetBytes(headerJson));

string payloadB64 = Base64UrlEncode(Encoding.UTF8.GetBytes(payloadJson));

string unsigned = $"{headerB64}.{payloadB64}";

using var hmac = new HMACSHA256(keyBytes);

byte[] signature = hmac.ComputeHash(Encoding.UTF8.GetBytes(unsigned));

string signatureB64 = Base64UrlEncode(signature);

return $"{unsigned}.{signatureB64}";

}

static bool ValidateToken(string token, byte[] keyBytes, string issuer, string audience, out string payloadJson, out string error)

{

payloadJson = null;

error = null;

var parts = token.Split('.');

if (parts.Length != 3) { error = "Invalid format"; return false; }

string unsigned = $"{parts[0]}.{parts[1]}";

string signature = parts[2];

using var hmac = new HMACSHA256(keyBytes);

string expectedSig = Base64UrlEncode(hmac.ComputeHash(Encoding.UTF8.GetBytes(unsigned)));

if (!CryptoEquals(expectedSig, signature))

{

error = "Invalid signature";

return false;

}

string payloadStr = Encoding.UTF8.GetString(Base64UrlDecode(parts[1]));

payloadJson = payloadStr;

if (!payloadStr.Contains($"\"iss\":\"{issuer}\"") || !payloadStr.Contains($"\"aud\":\"{audience}\""))

{

error = "Issuer or audience mismatch";

return false;

}

int expIndex = payloadStr.IndexOf("\"exp\":");

if (expIndex < 0)

{

error = "Expiration missing";

return false;

}

int start = expIndex + 6;

int end = payloadStr.IndexOfAny(new[] { ',', '}', ' ' }, start);

string expStr = payloadStr.Substring(start, end - start);

if (!long.TryParse(expStr, out long exp))

{

error = "Invalid expiration";

return false;

}

if (DateTimeOffset.UtcNow.ToUnixTimeSeconds() >= exp)

{

error = "Token expired";

return false;

}

return true;

}

static string Base64UrlEncode(byte[] input)

{

return Convert.ToBase64String(input).TrimEnd('=').Replace('+', '-').Replace('/', '\_');

}

static byte[] Base64UrlDecode(string input)

{

string padded = input.Replace('-', '+').Replace('\_', '/');

switch (padded.Length % 4)

{

case 2: padded += "=="; break;

case 3: padded += "="; break;

case 1: padded += "==="; break;

}

return Convert.FromBase64String(padded);

}

static bool CryptoEquals(string a, string b)

{

if (a.Length != b.Length) return false;

int result = 0;

for (int i = 0; i < a.Length; i++)

result |= a[i] ^ b[i];

return result == 0;

}

}





