Symmetrisk

Kryptering

Symmetrisk kryptering i .NET

Dette afsnit og i øvelserne præsenteres de symmetriske kryptografiske primitiver som er understøttet i .NET 7. Alle klasser relateret til kryptografi er indeholdt i **`System.Security.Cryptography`** namespace. Kryptografiens historie i Microsofts udviklingsmiljøer startede i 1996 med Win32 **`Cryptography API`** (Application Programming Interface) også kendt som Microsoft CryptoAPI. I tidligere versioner af .NET så man klasser, der havde navne, der sluttede på **`CryptoServiceProvider`**, og disse klasser var faktisk indpakning/wrappers over eksisterende kode fra Win32 Cryptography API. Andre klassenavne sluttede på **`Managed`**, og disse var managed kode skrevet specifikt til .NET frameworket.

I .NET 7 fortsætter man med at bygge videre på dette robuste fundament, og de ældre **`CryptoServiceProvider`** klasser er gradvist blevet erstattet eller suppleret med mere moderne og fleksible implementeringer. Kryptografi support i .NET er stadig robust og tilbyder alle de grundlæggende byggesten, der er nødvendige til applikationer der behandler følsomt data.

I .NET 7 får du out-of-the-box brugervenlige implementeringer til symmetriske krypteringsfunktioner (AES, TripleDES, osv.), hash-funktioner (SHA-1, SHA-256, SHA-384, SHA-512), keyed hash funktioner (HMAC med forskellige algoritmer som SHA-256, SHA-384, SHA-512), offentlig-nøglekryptering eller digital signatur (RSA, ECDSA) og PRNG'er.

Symmetriske algoritmer, egenskaber og metoder

AES (Advanced Encryption Standard): AES er en symmetrisk krypteringsalgoritme, der blev etableret af U.S. National Institute of Standards and Technology (NIST) i 2001. AES har en fast blokstørrelse på 128 bits og nøglestørrelser på 128, 192 eller 256 bits. Algoritmen er baseret på forskellige permutationer og transformationer, der følger en runde-baseret struktur.

DES (Data Encryption Standard): DES er en tidligere anvendt symmetrisk krypteringsalgoritme, der blev indført i 1977 af U.S. National Bureau of Standards (det nuværende NIST). DES bruger en 56-bit nøgle og har en blokstørrelse på 64 bits. På grund af den relativt lille nøglestørrelse og fremskridt inden for computerkraft, betragtes DES nu for at være usikker for mange applikationer.

Triple DES (3DES): Triple DES blev udviklet som en forbedret og sikrere version af DES. I stedet for en enkelt 56-bit nøgle, bruger Triple DES tre 56-bit nøgler, hvilket giver en samlet nøglestørrelse på 168 bits. Algoritmen krypterer data tre gange, derfor navnet "Triple DES".

AesGcm

AesGcm-klassen i .NET 7. Dette er en moderne klasse, der giver en effektiv implementering af AES-algoritmen med Galois/Counter Mode (GCM)[1], som giver både kryptering og autentificering.

Først skal der oprettes en ny instans af AesGcm-klassen med en tilfældig nøgle. AesGcm understøtter nøglelængder på 128, 192 og 256 bits. Her vil vi bruge en 256-bit nøgle:

```
byte[] key = new byte[32]; // 256 bits
RandomNumberGenerator.Fill(key);
var aes = new AesGcm(key);
```

[1] using System.Security.Cryptography;

Her anvendes `RandomNumberGenerator.Fill`-metoden til at generere en tilfældig nøgle. Dette er en anden sikker måde at generere kryptografiske nøgler på.
Nu kan vi kryptere en besked. AesGcm bruger en "`nonce`" (nummer, der bruges én gang) på 12 bytes. Vi skal også bruge en buffer til det krypterede output og et tag.

```
byte[] plaintext = Encoding.UTF8.GetBytes("Hello, world!");
byte[] nonce = new byte[AesGcm.NonceByteSizes.MaxSize];
byte[] ciphertext = new byte[plaintext.Length];
byte[] tag = new byte[AesGcm.TagByteSizes.MaxSize];

RandomNumberGenerator.Fill(nonce);
aes.Encrypt(nonce, plaintext, ciphertext, tag);
```

Her har vi konverteret en streng til et byte-array ved hjælp af Encoding.UTF8.GetBytes, genereret en tilfældig nonce, og brugt aes.Encrypt-metoden til at kryptere plaintexten.

Vi kan nu dekryptere ciphertexten tilbage til plaintext:

```
byte[] decryptedPlaintext = new byte[ciphertext.Length];
aes.Decrypt(nonce, ciphertext, tag, decryptedPlaintext);

string decryptedMessage = Encoding.UTF8.GetString(decryptedPlaintext);
Console.WriteLine(decryptedMessage); // Skriver "Hello, world!" til konsollen
```

Her har vi brugt **aes.Decrypt**-metoden til at dekryptere ciphertexten. Bemærk, at vi skal bruge den samme nonce og det samme tag, som vi brugte til at kryptere dataene. Derefter konverterer vi det dekrypterede byte-array tilbage til en streng med **Encoding.UTF8.GetString**.

Dette er bare en simpel demonstration af, hvordan man kan bruge AesGcm til symmetrisk kryptering i .NET. I en reel applikation skal du håndtere nøgle- og nonce-håndtering mere omhyggeligt, og du skal sørge for korrekt fejlhåndtering.

Der er vedhæftet eksempler på implementering af ”gammeldags” AES, DES og TripleDES vedhæftet under denne opgave på moodle.

I eksempler anvendes:

1. **byte[] encrypted:** Dette er den variable, der holder den krypterede version af den oprindelige streng. Når du krypterer en streng (eller nogen data), bliver output normalt binære data, så det er mest hensigtsmæssigt at gemme det som et byte-array.
2. **Key:** I kryptering er en nøgle en streng af tegn, der bruges til at kryptere og dekryptere data. Størrelsen og formatet af nøglen afhænger af den specifikke krypteringsalgoritme, du bruger. For eksempel bruger AES en nøgle, der er 128, 192, eller 256 bit lang.
3. **IV (Initialiseringsvektor):** Dette er en tilfældig streng, der bruges som en slags "startværdi" for krypteringsalgoritmen. IV gør det muligt for to stykker tekst, der er identiske og krypteret med den samme nøgle, at have forskellige krypterede repræsentationer. Dette forbedrer

sikkerheden. IV er ikke hemmelig og kan sikkert sendes sammen med den krypterede data, men den skal være unik for hver enkelt besked.

4. **Padding:** Dette er teknikken til at fylde input data op til en vis længde, så det passer til krypteringsalgoritmens krav. For eksempel kræver AES, at input data er en multiplum af 16 byte. Hvis dataene du ønsker at kryptere ikke opfylder dette krav, kan du bruge padding til at fylde den op til den nødvendige længde.
5. **ICryptoTransform:** Dette er en interface i .NET, der repræsenterer en generel kryptografisk transformation, som kan være kryptering, dekryptering, eller andre operationer. Når du kalder **CreateEncryptor** eller **CreateDecryptor** på en krypteringsalgoritme, får du en ICryptoTransform, som du kan bruge til at udføre transformationen.
6. **MemoryStream:** Dette er en klasse i .NET, der repræsenterer en strøm af data, der gemmes i hukommelsen (i modsætning til på disk eller over netværket). Det er nyttigt, når du skal arbejde med data i en strøm-lignende måde (for eksempel, læse fra eller skrive til det lidt ad gangen), men du ikke har brug for eller ønsker at arbejde med en fil eller netværksforbindelse. I dette tilfælde bruges det til at opsamle de krypterede eller dekrypterede data, så de kan returneres fra funktionen.

Øvelse

Skriv et C# -program, der giver en bruger mulighed for at vælge en symmetrisk krypterings algoritme fra en ComboBox eller via konsol, generer nøgler, krypter og dekrypter beskeder. Vis plain text cipher text både i ASCII og HEX og det samme nøglerne og IV; vis også den tid, der kræves af krypterings- og dekrypteringsoperationerne. Et forslag til en grænseflade er vist nedenfor, men du er velkommen til at udvikle en anden.