Hashing

Kryptering

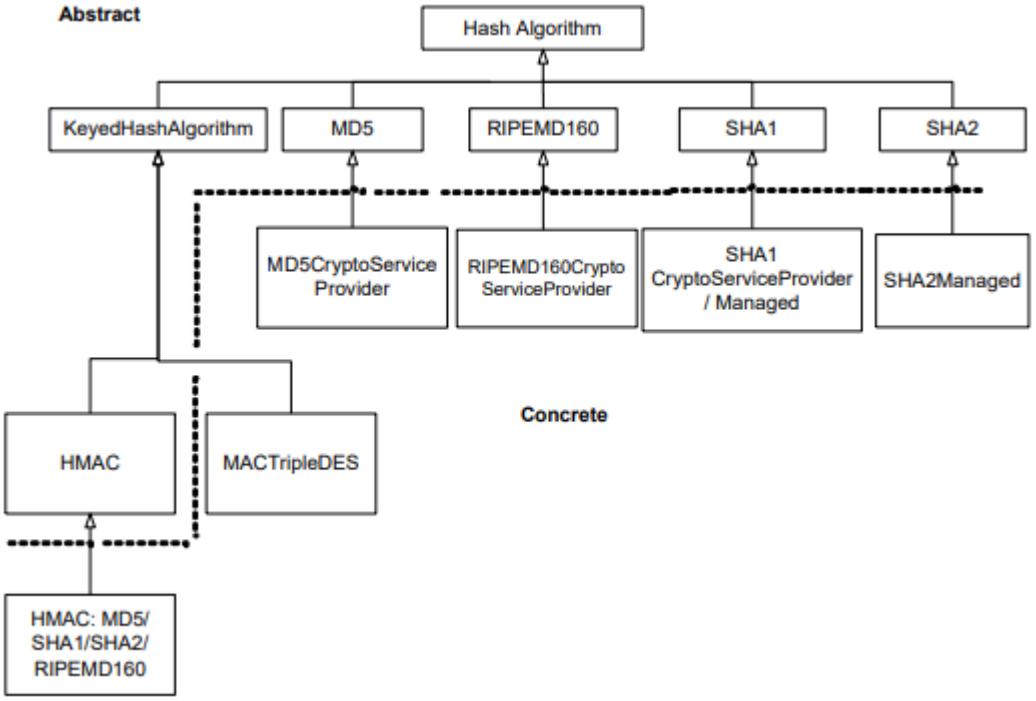
Hash funktioner og MAC koder i .NET

I denne øvelse skal du arbejde med hashfunktionerne og den afledte funktionalitet Message Authentication Codes (MAC), der understøttes af .NET-framework. Yderligere skal der gøres brug at sikre tilfældigheds-talgeneratorer.

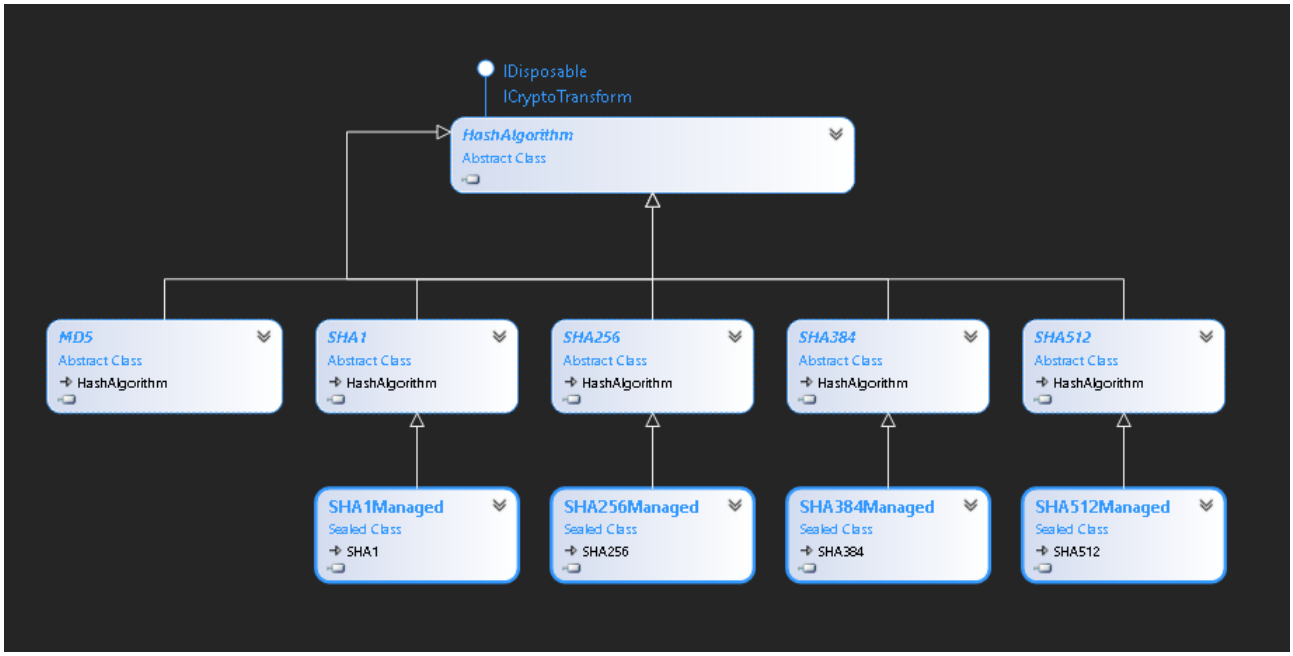
.NET-frameworket understøtter de forældet, men stadig i vid udstrækning anvendte MD5 (128 bit) og SHA1 (160 bit). Udover SHA1 og MD5 understøttes den nuværende standard SHA2 i alle tre outputstørrelser 256, 384 og 512 bit og den mindre hyppig RIPEMD (160 bit).

MAC'er (Message Authentication Codes) også kaldet keyed hash functions fordi de er bygget ud fra en hash-funktion der benytter en hemmelig nøgle. Men der er undtagelser fra denne regel, undtagelser opstår er når MAC-codes, inkorporeres i symmetrisk krypteringsfunktioner (kommer senere).

En af de MAC codes, der understøttes af .NET, er HMAC, som er en keyed hash function og foretrukket fordi, den kan anvendes på en hvilken som helst af de hashfunktioner, der er tilgængelige i .NET: MD5, SHA1 og SHA2. Nedenstående figur[i] viser klasseorganisationen for hash-algoritmer og MAC-koder, bemærk sondringen mellem abstrakte og konkrete klasser.:



Figur 1: klasseorganisationen for hash-algoritmer



Figur 2: Afledte klasser af HashAlgorithm Class i .NET Core

Hash funktioner

Hash-funktioner er afledt af den abstrakte klasse HashAlgoritme, der findes i System.Security.Cryptography namespace. Nogle egenskaber og metoder er skitseret i tabel 1 og 2.

Property	Get/Set	Type	Brief Description
`InputBlockSize`	g	Int	Bit size of the input block, returns 1 unless overwritten
`OutputBlockSize`	g	Int	Bit size of the output block, returns 1 unless overwritten
`HashSize`	g	Int	Bit size of the hash
`Hash`	g	Byte[]	Value of the hash

Tabel 1: nogle egenskaber til hashfunktioner i .NET

Method	Return type	Brief Description
`Create()`	HashAlgorithm	Creates the object (SHA1 is the default instance) <<- **Obsolete**
`Create(String)`	HashAlgorithm	Creates the object with the string specifying the name of the particular implementation given as string (MD5, SHA1, etc.) <<- **Obsolete**
`ComputeHash(Byte[])`	Byte[]	Computes the hash from a byte array
`ComputeHash(Stream)`	Byte[]	Computes the hash from a stream object
`ComputeHash(Byte[], Int32, Int32)`	Byte[]	Computes the hash from a specific region of a byte array
`TransformBlock(byte[] inputBuffer, int inputOffset, int inputCount, byte[] outputBuffer, int outputOffset)`	Int	Computes the hash of a specified region of a byte array and copies the region to the specified region of the output byte array. Return the number of bytes written.
`TransformFinalBlock(byte[] inputBuffer, int inputOffset, int inputCount`	Byte[]	Computes the hash of a specified region of a byte array, returns a copy a of the part of the input that is hashed

Tabel 2: nogle metoder til hashfunktioner i .NET

For at beregne hash fra en/et byte-array eller -stream kan du blot kalde til `ComputeHash`-metoden som skitseret i ovenstående tabel. I dette nedenstående eksempel 1 benyttes også et `RandomNumberGenerator` objekt til at generere nogle vilkårlige tal, der er senere, bliver hashed. For at generere tilfældige værdier skal du blot oprette et `RandomNumberGenerator` objekt og kalde `GetBytes` metoden på et specifikt byte array.

```
// Generer 16 tilfældige bytes
byte[] randomBytes = new byte[16];
using (RandomNumberGenerator rng = RandomNumberGenerator.Create())
{
    rng.GetBytes(randomBytes);
}

Console.WriteLine("Random bytes: " + BitConverter.ToString(randomBytes));

// Beregn hash-værdien af de tilfældige bytes
byte[] hashValue;
using (SHA256 sha256 = SHA256.Create())
{
    hashValue = sha256.ComputeHash(randomBytes);
}

Console.WriteLine("Hash value: " + BitConverter.ToString(hashValue));
```

Eksempel 1: generering af nogle tilfældige bytes og beregning af deres hash

Eksempel 2 viser, hvordan man beregner hash for en given fil, dette er en ofte anvendte procedure for at kontrollere filernes integritet eller for at sammenligne to filer (eller objekter) er identiske, da kun identiske objekter kan hash til den samme værdi (forudsat at hash-funktionen er kollisionsfri).

```
string filePath = "path/to/your/file.txt";

using (FileStream stream = File.OpenRead(filePath))
{
    using (SHA256 sha256 = SHA256.Create())
    {
        byte[] hashValue = sha256.ComputeHash(stream);
        string hashString = BitConverter.ToString(hashValue).Replace("-", String.Empty);

        Console.WriteLine($"Hash value for file {filePath}: {hashString}");
    }
}
```

Eksempel 2: Eksempel til beregning af hash fra en given fil

Key hash funktioner

.NET framework giver os implementering af HMAC-keyed hash funktion. Egenskaberne og metoderne for KeyedHashAlgorithm objekter er næsten identisk med HashAlgorithm (en klasse, som de arver). Den eneste ekstra egenskab, er den, der skal get/set nøglen som beskrevet i nedenstående.

Property	Get/Set	Type	Brief Description
`Key`	g/s	Byte[]	Value of the key for the HMAC

Tabel 3: nøgle egenskaben for keyed hash-algoritmer i .NET

I eksempel 3 og 4 ses, hvordan man kan instantiere en HMAC med en bestemt hash funktion, hvordan man genererer autentifikation med `ComputeMAC(byte[] mes, byte[] key)` og derefter kontrolleres det med `CheckAuthenticity(byte[] mes, byte[] mac, byte[] key))`.

```
private HMAC myMAC;
0 references | 0 changes | 0 authors, 0 changes
public MACHandler(string name)
{
    if (name.CompareTo("SHA1") == 0)
    {
        myMAC = new System.Security.Cryptography.HMACSHA1();
    }
    if (name.CompareTo("MD5") == 0)
    {
        myMAC = new System.Security.Cryptography.HMACMD5();
    }
    if (name.CompareTo("RIPEMD") == 0)
    {
        myMAC = new System.Security.Cryptography.HMACRIPEMD160();
    }
    if (name.CompareTo("SHA256") == 0)
    {
        myMAC = new System.Security.Cryptography.HMACSHA256();
    }
    if (name.CompareTo("SHA384") == 0)
    {
        myMAC = new System.Security.Cryptography.HMACSHA384();
    }
    if (name.CompareTo("SHA512") == 0)
    {
        myMAC = new System.Security.Cryptography.HMACSHA512();
    }
}
```

Eksempel 3: oprettelse af et HMAC-objekt med en bestemt hash-funktion

```
public bool CheckAuthenticity(byte[] mes, byte[] mac, byte[] key)
{
    myMAC.Key = key;
    if (CompareByteArrays(myMAC.ComputeHash(mes), mac, myMAC.HashSize / 8) == true)
    {
        return true;
    }
    else
    {
        return false;
    }
}
0 references | 0 changes | 0 authors, 0 changes
public byte[] ComputeMAC(byte[] mes, byte[] key)
{
    myMAC.Key = key;
    return myMAC.ComputeHash(mes);
}
0 references | 0 changes | 0 authors, 0 changes
public int MACByteLength()
{
    return myMAC.HashSize / 8;
}
1 reference | 0 changes | 0 authors, 0 changes
private bool CompareByteArrays(byte[] a, byte[] b, int len)
{
    for (int i = 0; i < len; i++)
        if (a[i] != b[i]) return false;
    return true;
}
```

Eksempel 4: beregner HMAC og bekræfter derefter ægtheden af en meddelelse

Hash funktioner og MAC-koder som cryptostreams

Det sidste trick, der kan være nyttigt at vide, er, at du kan ”pass” hashfunktioner eller HMAC'er som transformationer integreret i CryptoStreams. I eksempel 5 viser, hvordan CryptoStreams fungere.

I eksempel 5 vises, hvordan du kan bruge **HMACSHA256** og **SHA256** sammen med **CryptoStream** i .NET. Dette eksempel viser, hvordan du kan beregne en SHA-256 hash af en besked og en HMACSHA256 værdi ved hjælp af **CryptoStream**.

```
string message = "Your message here";
byte[] key = Encoding.UTF8.GetBytes("SuperSecretKey"); // Secret key for HMAC

// Convert the message to a byte array
byte[] messageBytes = Encoding.UTF8.GetBytes(message);

// Compute SHA256 hash
byte[] hashValue;
using (SHA256 sha256 = SHA256.Create())
{
    using (MemoryStream ms = new MemoryStream(messageBytes))
    {
        using (CryptoStream cs = new CryptoStream(ms, sha256, CryptoStreamMode.Read))
        {
            // Read the CryptoStream to compute the hash
            cs.CopyTo(Stream.Null);
        }
    }
    hashValue = sha256.Hash;
}

Console.WriteLine("SHA256 Hash: " + BitConverter.ToString(hashValue));

// Compute HMACSHA256
byte[] hmacValue;
using (HMACSHA256 hmac = new HMACSHA256(key))
{
    using (MemoryStream ms = new MemoryStream(messageBytes))
    {
        using (CryptoStream cs = new CryptoStream(ms, hmac, CryptoStreamMode.Read))
        {
            // Read the CryptoStream to compute the HMAC
            cs.CopyTo(Stream.Null);
        }
    }
    hmacValue = hmac.Hash;
}

Console.WriteLine("HMACSHA256: " + BitConverter.ToString(hmacValue));
```

Eksempel 5: eksempel på HMACSHA256 og SHA256 anvendt i CryptoStreams

Her anvendes vi **CryptoStream** sammen med **MemoryStream** til at beregne både SHA-256 hash og HMACSHA256 værdier af en besked. **MemoryStream** bruges til at omdanne beskeden til en stream, som **CryptoStream** kan læse, og hashen eller HMAC beregnes, når vi læser streamen.

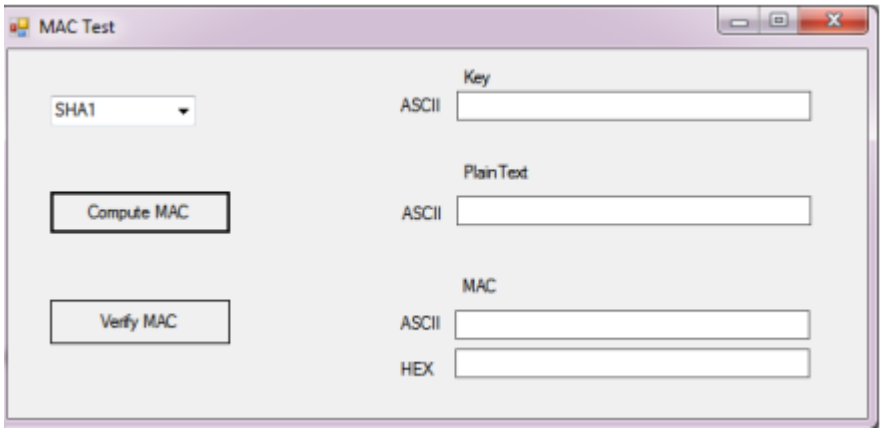
Dette giver en mere strømlinet måde at arbejde med hashing og HMAC, især hvis du arbejder med større mængder data, da det kan læses og behandles som en stream.

Øvelser

Skriv et C# -program, der giver en bruger mulighed for at vælge en Hash- eller HMAC-algoritme fra en Combo Box eller via konsol, generer nøgler (i tilfælde af HMAC), hash besked og verificer (i tilfælde af HMAC) deres hashes.

Vis plain text og hash både i ASCII og HEX; vis også den tid, der kræves af hash- og HMAC-operationerne. Et forslag til start kunne være grænsefladen som er vist nedenfor, men du er velkommen til at udvikle en anden. Resultaterne af tidtagningen skal præsenteres i en tabelform som vist på næste side.

Der kan læses om hash Ripemd algoritmen https://nciphers.com/tutorial/ripemd128/



[i] https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.hashalgorithm?view=netframework-7.0