```
        Console.WriteLine("---------------------------------");
        Console.WriteLine();
        Console.WriteLine("Original Message 1 : " + originalMessage);
        Console.WriteLine("Original Message 2 : " + originalMessage2);
        Console.WriteLine();

        var sha1hashedMessage =
HashData.ComputeHashSHA1(Encoding.UTF8.GetBytes(originalMessage));
        var sha1hashedMessage2 =
HashData.ComputeHashSHA1(Encoding.UTF8.GetBytes(originalMessage2));

        var sha256hashedMessage =
HashData.ComputeHashSHA256(Encoding.UTF8.GetBytes(originalMessage));
        var sha256hashedMessage2 =
HashData.ComputeHashSHA256(Encoding.UTF8.GetBytes(originalMessage2));

        var sha512hashedMessage =
HashData.ComputeHashSHA512(Encoding.UTF8.GetBytes(originalMessage));
        var sha512hashedMessage2 =
HashData.ComputeHashSHA512(Encoding.UTF8.GetBytes(originalMessage2));

        Console.WriteLine();
        Console.WriteLine("SHA 1 Hashes");
        Console.WriteLine();
        Console.WriteLine("Message 1 hash = " +
Convert.ToBase64String(sha1hashedMessage));
        Console.WriteLine("Message 2 hash = " +
Convert.ToBase64String(sha1hashedMessage2));
        Console.WriteLine();

        Console.WriteLine("SHA 256 Hashes");
        Console.WriteLine();
        Console.WriteLine("Message 1 hash = " +
Convert.ToBase64String(sha256hashedMessage));
        Console.WriteLine("Message 2 hash = " +
Convert.ToBase64String(sha256hashedMessage2));
        Console.WriteLine();
        Console.WriteLine("SHA 512 Hashes");
        Console.WriteLine();
        Console.WriteLine("Message 1 hash = " +
Convert.ToBase64String(sha512hashedMessage));
        Console.WriteLine("Message 2 hash = " +
Convert.ToBase64String(sha512hashedMessage2));
        Console.WriteLine();
        Console.ReadLine();
    }
```

Code Listing 6

Message Authentication Codes

If you combine a one-way hash function with a secret cryptographic key, you get what is called a hash message authentication code (HMAC).

Like a hash code, an HMAC is used to verify the integrity of a message. An HMAC also allows you to verify the authentication of that message because only a person who knows the key can calculate the same hash of the message. An HMAC can be used with different hashing functions like MD5 or the SHA family of algorithms.

The cryptographic strength of an HMAC depends on the size of the key that is used. The most common attack against an HMAC is a brute force attack to uncover the key. HMACs are substantially less affected by collisions than their underlying hashing algorithms alone.

HMACs are used when you need to check both integrity and authenticity. For example, consider a scenario in which you are sent a piece of data along with its hash. You can verify the integrity of the message by re-computing the hash of the message and comparing it with the hash that you received. However, you don't know for sure if the message and the hash was sent by someone you know or trust. If you used an HMAC, you could re-compute the HMAC by using a secret key that only you and a trusted party know, and compare it with the HMAC you just received. This serves the purpose of authenticity.

In the following example, I will show you how to use an HMAC that uses a 32-byte (256-bit) key and an HMAC based on the SHA-256 hashing algorithm. I used a 256-bit key here as it is the same size key as you would use for AES encryption but you can go higher. In addition to **HMACSHA256**, you can use **HMACSHA1**, **HMACSHA512**, and **HMACMD5**. The interfaces are all the same.

Let’s look at the following code.

```
public class HMAC
{
    private const int KEY_SIZE = 32;

    public static byte[] GenerateKey()
    {
        using (var randomNumberGenerator = new RNGCryptoServiceProvider())
        {
            var randomNumber = new byte[KEY_SIZE];
            randomNumberGenerator.GetBytes(randomNumber);

            return randomNumber;
        }
    }

    public static byte[] ComputeHMACSHA256(byte[] toBeHashed, byte[] key)
    {
        using (var hmac = new HMACSHA256(key))
        {
            return hmac.ComputeHash(toBeHashed);
        }
    }

    public static byte[] ComputeHMACSHA1(byte[] toBeHashed, byte[] key)
    {
        using (var hmac = new HMACSHA1(key))
        {
            return hmac.ComputeHash(toBeHashed);
        }
    }

    public static byte[] ComputeHMACSHA512(byte[] toBeHashed, byte[] key)
    {
        using (var hmac = new HMACSHA512(key))
        {
            return hmac.ComputeHash(toBeHashed);
        }
    }

    public static byte[] ComputeHMACMD5(byte[] toBeHashed, byte[] key)
    {
        using (var hmac = new HMACMD5(key))
        {
            return hmac.ComputeHash(toBeHashed);
        }
    }
}
```

Code Listing 7

The hash class contains a method to generate the key by using the **`RNGCryptoServiceProvider`** that we have previously discussed. Then, there is the **`ComputeHmacSHA256`** method that takes a byte array of the code that you want to hash and a byte array for the encryption key.

To use the code, you do the following.

```
class Program
{
    static void Main(string[] args)
    {
        var originalMessage = "Original Message to hash";
        var originalMessage2 = "This is another message to hash";

        Console.WriteLine("HMAC Demonstration in .NET");
        Console.WriteLine("--------------------------");
        Console.WriteLine();

        var key = HMAC.GenerateKey();

        var hmacMd5Message =
HMAC.ComputeHMACMD5(Encoding.UTF8.GetBytes(originalMessage), key);
```