When you hash some data, the hash will be the same every time you perform the operation unless the original data changes in some way. Even if the data only changes by one bit, the resulting hash code will be completely different. This makes hashing the perfect mechanism for checking the integrity of data.

This is useful when you want to send data across a network to another recipient. Before sending the data, you calculate a hash of the data to get its unique fingerprint. You then send that data and the hash to the recipient. They then calculate the hash of the data they have just received and then compare it to the hash you sent. If the hash codes are the same, then the data has been successfully received without data loss or corruption. If the hash codes do not match, then the data received is not the same as the data originally sent.

The two most common hashing methods used are MD5 and the SHA family of hashes (SHA-1, SHA-256, and SHA-512).

MD5

The MD5 message digest algorithm is a widely used cryptographic hash function that produces a 128-bit (16-byte) hash value, which is typically expressed in text format as a 32-digit hexadecimal number or as a Base64-encoded string. MD5 has been used in a wide number of cryptographic applications and is also commonly used to verify file integrity.

MD5 was designed by Ron Rivest in 1991 to replace MD4, an earlier hash function. In 1996, a flaw was found in the design of MD5. While it was not deemed a fatal weakness at the time, cryptographers began to recommend the use of other algorithms such as the SHA family. In 2004, it was shown that MD5 is not collision-resistant, which means that it is possible that generating an MD5 of two sets of data could result in the same hash (although this is quite rare).

You may still need to use MD5 in your applications if you are checking the integrity of data coming from a legacy system that makes use of MD5.

It is easy to calculate an MD5 hash of some data in .NET as shown in the following code.

```
public class HashData
{
    public static byte[] ComputeHashMD5(byte[] toBeHashed)
    {
        using (var md5 = MD5.Create())
        {
            return md5.ComputeHash(toBeHashed);
        }
    }
}
```

Code Listing 3

You first call **`MD5.Create()`** and then call **`ComputeHash()`** on the object while passing in the data you want to be hashed. The input to **`ComputeHash()`** has to be a byte array, so if you are calculating a hash of a file, you will need to load it up into a byte array first.