A password-based key derivation function, or PBKDF2 as it is also known, is part of the RSA Public Key Cryptographic Standards series (PKCS #5 version 2.0). PBKDF2 is also part of the Internet Engineering Task Force's RFC2898 specification. A password-based key derivation function takes a password, a salt to add additional entropy to the password, and a number of iterations value. The number of iterations value repeats a hash or encryption cipher over the password multiple times to produce a derived key for the password that can be stored in a database or used as an encryption key.

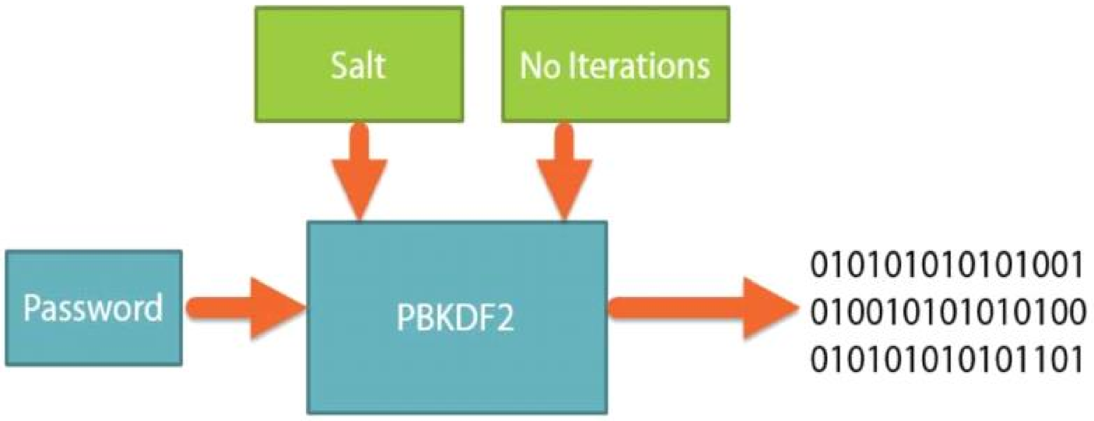

Figure 9: Password-based key derivation function

By repeating a hash or encryption process over the password multiple times, you are algorithmically slowing down the hashing process which makes brute force attacks against the password much harder. This means that fewer passwords can be tested at once. As processors get faster over time, you can increase the number of iterations used to create the new password, which means a password-based key derivation function can scale with Moore's law. A good default to start with for the number of iterations is around 50,000.

By adding a salt to the derivation function, you further reduce the ability of a rainbow table to be used to recover the original password. A good minimum length for your salt is at least 64 bits (8 bytes).

The .NET object we will use to perform the key derivation is **`Rfc2898DeriveBytes`**. When constructing this object, you pass in the data to be hashed as a byte array, the salt as a byte array, and the number of rounds you want the algorithm to perform.

The salt is generated by using the **`RNGCryptoServiceProvider`** object. The salt does not have to be secret and can be stored alongside the hashed password. The salt is designed to give added entropy to the password being hashed.

The following code demonstrates how to hash a password with **`Rfc2898DeriveBytes`**.

```
public class PBKDF2
{
```

A password-based key derivation function, or PBKDF2 as it is also known, is part of the RSA Public Key Cryptographic Standards series (PKCS #5 version 2.0). PBKDF2 is also part of the Internet Engineering Task Force's RFC2898 specification. A password-based key derivation function takes a password, a salt to add additional entropy to the password, and a number of iterations value. The number of iterations value repeats a hash or encryption cipher over the password multiple times to produce a derived key for the password that can be stored in a database or used as an encryption key.

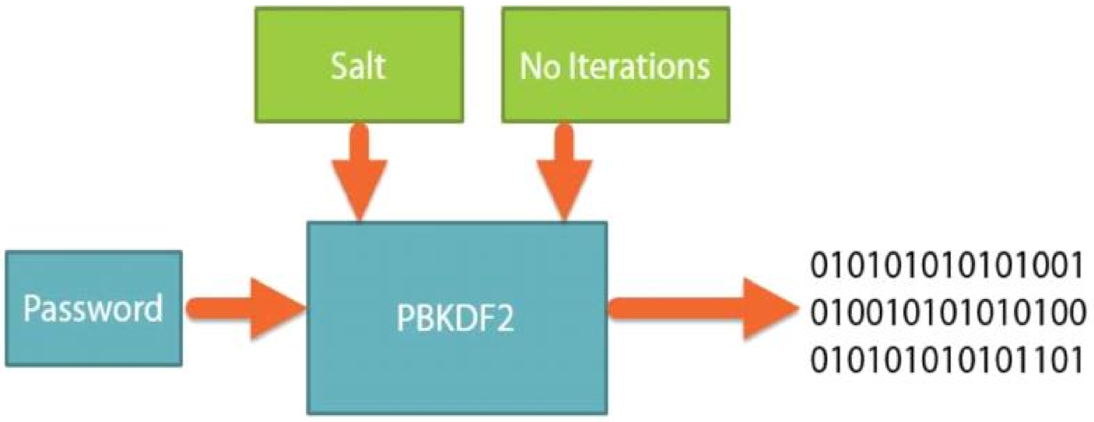

Figure 9: Password-based key derivation function

By repeating a hash or encryption process over the password multiple times, you are algorithmically slowing down the hashing process which makes brute force attacks against the password much harder. This means that fewer passwords can be tested at once. As processors get faster over time, you can increase the number of iterations used to create the new password, which means a password-based key derivation function can scale with Moore's law. A good default to start with for the number of iterations is around 50,000.

By adding a salt to the derivation function, you further reduce the ability of a rainbow table to be used to recover the original password. A good minimum length for your salt is at least 64 bits (8 bytes).

The .NET object we will use to perform the key derivation is **`Rfc2898DeriveBytes`**. When constructing this object, you pass in the data to be hashed as a byte array, the salt as a byte array, and the number of rounds you want the algorithm to perform.

The salt is generated by using the **`RNGCryptoServiceProvider`** object. The salt does not have to be secret and can be stored alongside the hashed password. The salt is designed to give added entropy to the password being hashed.

The following code demonstrates how to hash a password with **`Rfc2898DeriveBytes`**.

```
public class PBKDF2
{
```