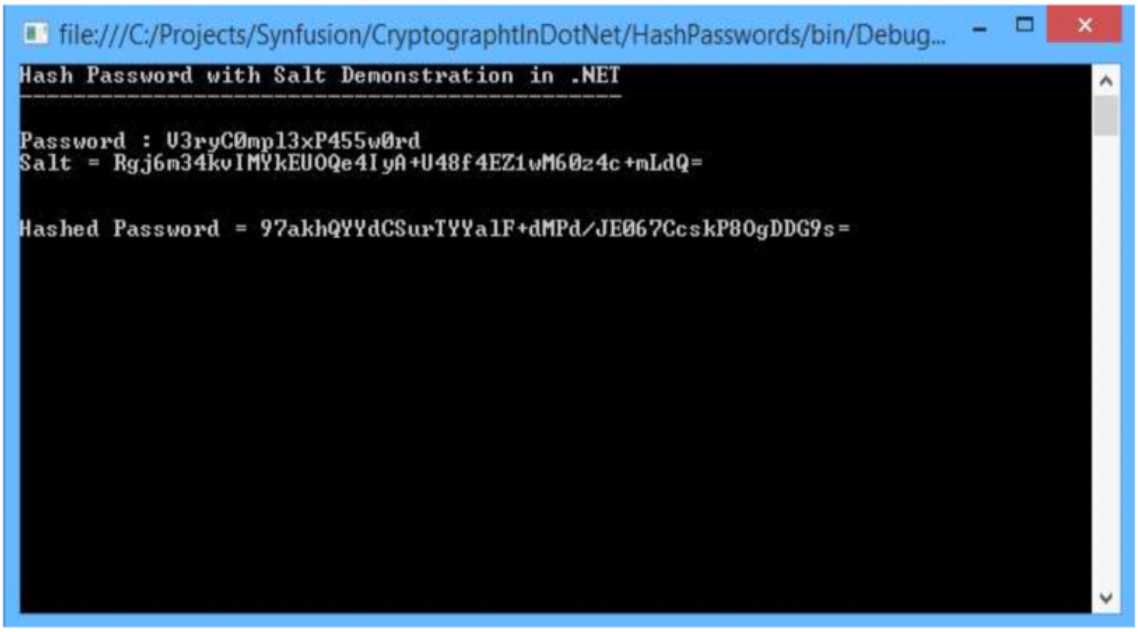

Figure 8: Example of a salted and hashed password

In this example, the salt value is combined with the password and then hashed by using SHA-256. When you store the password in the database, you also store the salt value. The salt does not need to be secret; it is there to add entropy to the password to make it harder to brute force attack or attack by using a rainbow table.

This solution for storing a password is much better than just storing a hashed password but it is still susceptible to brute force attacks as processor speeds scale with Moore's law. A stored hashed and salted password may be safe today but, in five years, it might be a trivial job to crack it.

A better solution is to use a password-based key derivation function to hash and store your password.

Password-based Key Derivation Functions

As we just discussed, the problem with hashing and storing a password (even with a salt) is that as processors get faster over the years, we run the risk of what we currently think are secure passwords being compromised because processors can use brute force attacks and rainbow table attacks faster. What we need is a solution that allows us to still hash our passwords but helps us guard against advancements in Moore's law.

Note: Moore's law, named after Gordon E. Moore, the co-founder of Intel, states that over time the number of transistors in a circuit will double approximately every 2 years.