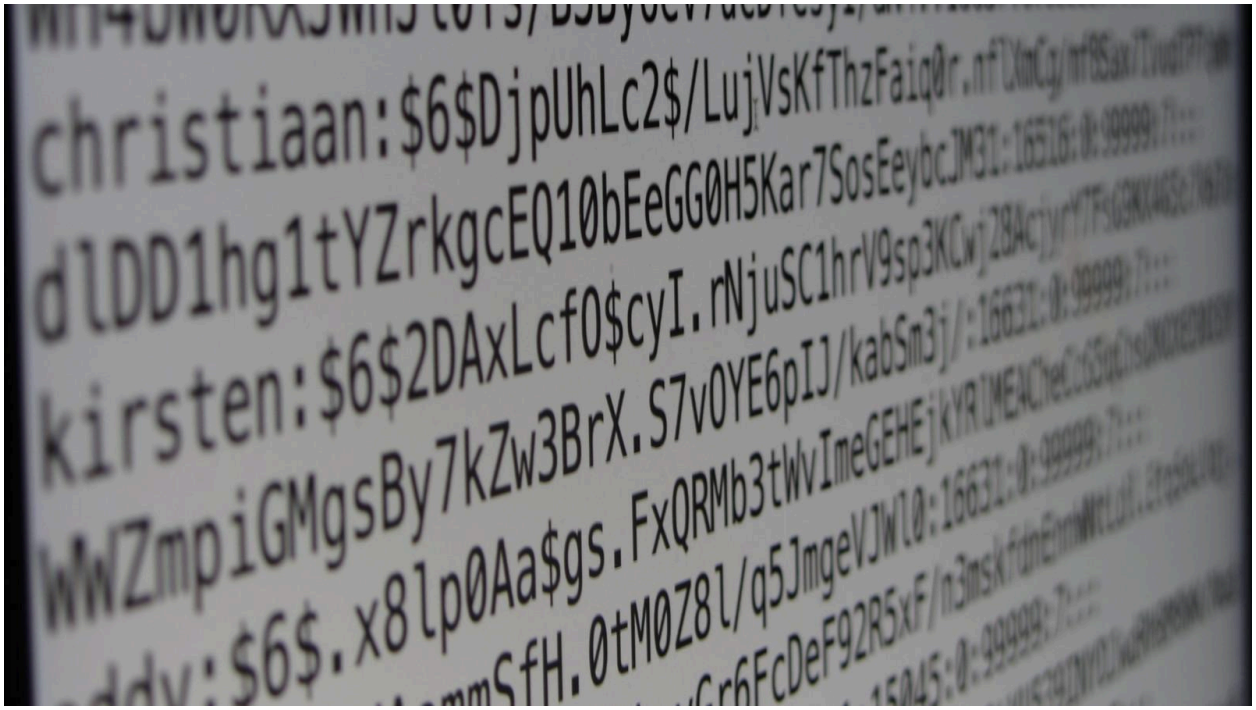


🔐 Salt & Encryption Tutorial

Protecting Passwords the Right Way

📌 Important Clarification

Before we start:

- **Encryption** → reversible (you can decrypt data)
- **Hashing** → one-way (you cannot reverse it)

👉 **Passwords should NOT be encrypted — they should be hashed with a salt**

🧂 What is a Salt?

A **salt** is a **random value added to a password before hashing**.

Why it matters:

Without salt:

password123 → same hash every time ❌

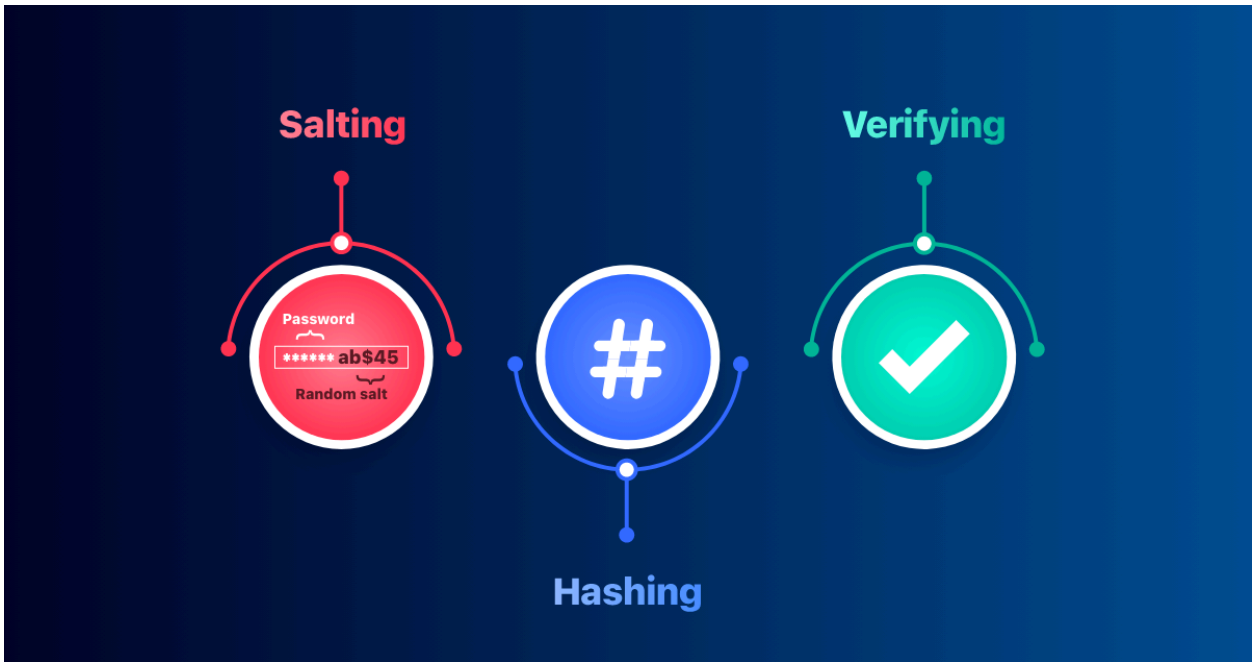
With salt:

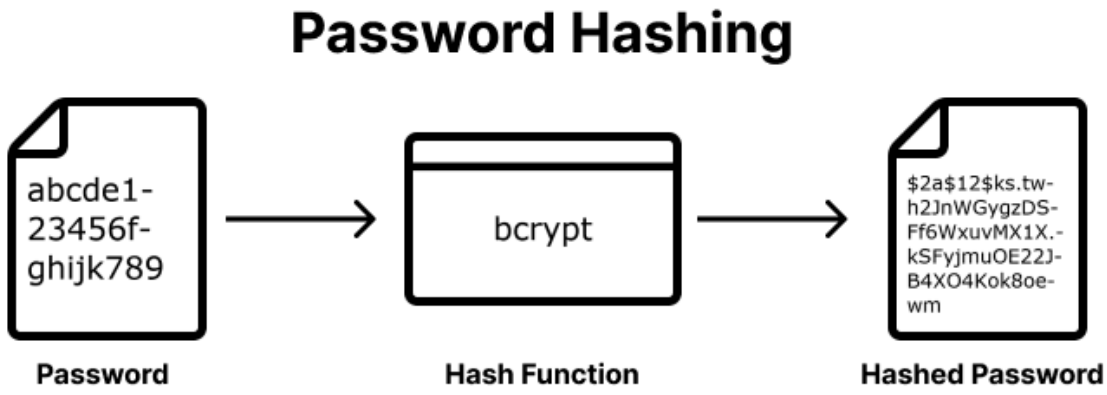
password123 + randomSalt → unique hash every time ✅

🎯 Why Use Salting?

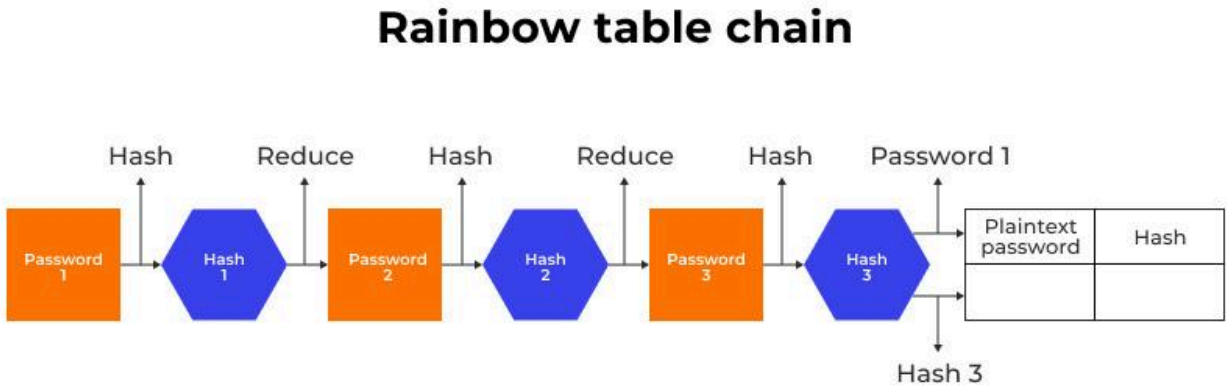
- Prevents **rainbow table attacks**
- Prevents attackers from spotting identical passwords
- Adds randomness to each password hash
- Strengthens security even if the database is leaked

🧠 How It Works (Concept)





wallarm



Process:

1. User enters password

2. System generates random salt
3. Combine password + salt
4. Hash the result
5. Store:
    - Hash
    - Salt

🔁 Authentication Process

1. User logs in
2. Retrieve stored salt
3. Combine input password + salt
4. Hash again
5. Compare hashes

🐍 Python Example (Secure Password Hashing)

Using `hashlib` + `os`

```
import os
import hashlib

def hash_password(password: str):
    salt = os.urandom(16)  # 16-byte random salt
    pwd_hash = hashlib.pbkdf2_hmac(
        'sha256',              # Hash algorithm
        password.encode(),      # Convert to bytes
        salt,                  # Salt
        100000                  # Iterations
    )
    return salt, pwd_hash

def verify_password(stored_salt, stored_hash, password_attempt):
    new_hash = hashlib.pbkdf2_hmac(
        'sha256',
        password_attempt.encode(),
```

```
        stored_salt,
        100000
    )
    return new_hash == stored_hash


# Example usage
salt, hashed = hash_password("MySecurePassword123")

print("Salt:", salt)
print("Hash:", hashed)

print("Verify:", verify_password(salt, hashed, "MySecurePassword123"))
```

🔒 Why This is Secure

- Uses **PBKDF2** (key stretching)
- Uses **100,000 iterations** (slows attackers)
- Uses **random salt per password**

💻 C# Example (.NET Secure Hashing)

Using `Rfc2898DeriveBytes` (PBKDF2)

```
using System;
using System.Security.Cryptography;

class Program
{
    static void Main()
    {
        string password = "MySecurePassword123";

        byte[] salt;
        byte[] hash;

        CreateHash(password, out salt, out hash);
```

```
        Console.WriteLine("Salt: " + Convert.ToBase64String(salt));
        Console.WriteLine("Hash: " + Convert.ToBase64String(hash));

        bool isValid = VerifyPassword(password, salt, hash);
        Console.WriteLine("Verified: " + isValid);
    }

    static void CreateHash(string password, out byte[] salt, out byte[] hash)
    {
        salt = RandomNumberGenerator.GetBytes(16);

        using (var pbkdf2 = new Rfc2898DeriveBytes(password, salt, 100000,
HashAlgorithmName.SHA256))
        {
            hash = pbkdf2.GetBytes(32);
        }
    }

    static bool VerifyPassword(string password, byte[] salt, byte[] storedHash)
    {
        using (var pbkdf2 = new Rfc2898DeriveBytes(password, salt, 100000,
HashAlgorithmName.SHA256))
        {
            byte[] newHash = pbkdf2.GetBytes(32);
            return CryptographicOperations.FixedTimeEquals(newHash, storedHash);
        }
    }
}
```

🔐 Best Practices (Very Important)

✅ Always Do This:

- Use **PBKDF2, bcrypt, scrypt, or Argon2**
- Use **random salt per user**
- Store **salt + hash together**
- Use **high iteration counts**

❌ Never Do This:

- Store plain text passwords
- Use MD5 or SHA1 ❌
- Reuse the same salt
- Skip salting

🧪 Real-World Example (Database Storage)

Username	Salt	Hash
user1	random123	abcxyz...
user2	random456	pqrs...

Even if both use the same password → hashes are different ✅

🧠 Cybersecurity Insight (Advanced)

This connects directly to your dissertation work:

- Salted hashes can be used in **behavioral anomaly detection**
- Sudden changes in authentication patterns may indicate:
  - Credential stuffing
  - USB-delivered malware attempting login
- Logging hash attempts (not passwords) supports **secure telemetry**

🧩 Practice Exercise

1. Modify the Python script to:
   - Store salt + hash in a file
2. Create a login function
3. Test with:
   - Correct password
   - Incorrect password

📌 Summary

- Salt adds randomness to passwords
- Hashing protects stored credentials
- Together they prevent common attacks
- Always use modern algorithms like PBKDF2
