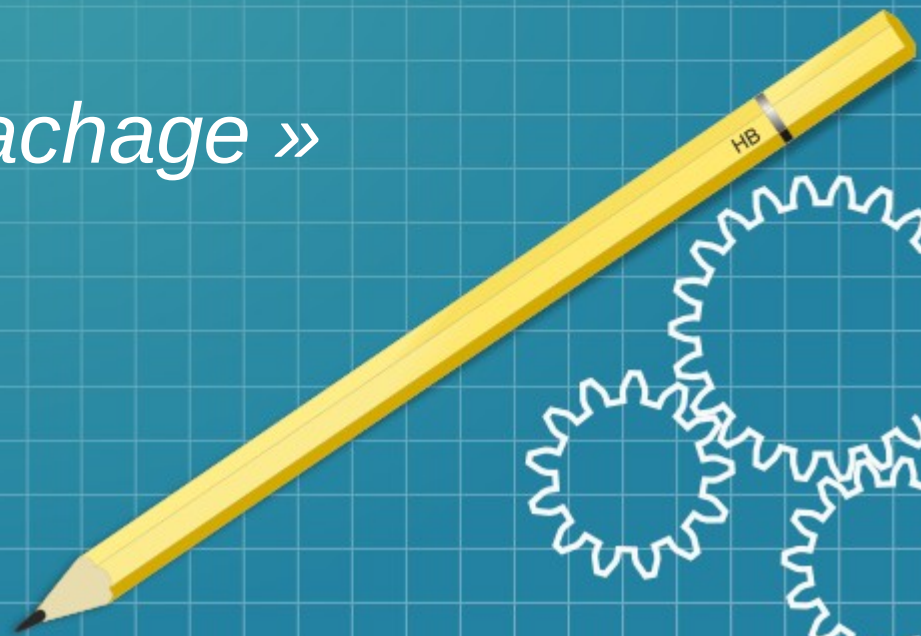
Politique d’entreprise :
le stockage de mot de passe

Principe du « hachage »



Mots de passe = données sensibles

Le principe de base pour un administrateur du système d'information est de rendre la récupération des mots de passe la plus difficile possible en cas d'intrusion dans la base de données.

- Le stockage des **mots de passe en clair est évidemment à exclure**...
- Le chiffrement des **mots de passe est à éviter** car cela supposerait que l'administrateur soit capable de faire l'inverse...

Fonctions de hachage

Le principe du hachage consiste à associer une donnée de taille fixe à une donnée de taille arbitraire par l'application d'une **fonction cryptographique** :

https://fr.wikipedia.org/wiki/Fonction_de_hachage_cryptographique

Cela sous-entend deux qualités importantes :

- Il est **impossible de retrouver le mot de passe à partir du hash**
- Deux mots de passe différents **ne peuvent donner le même hash**

→ Pour vérifier un mot de passe, on calcule le hash de celui entré et on le compare au hash stocké dans la base de données.

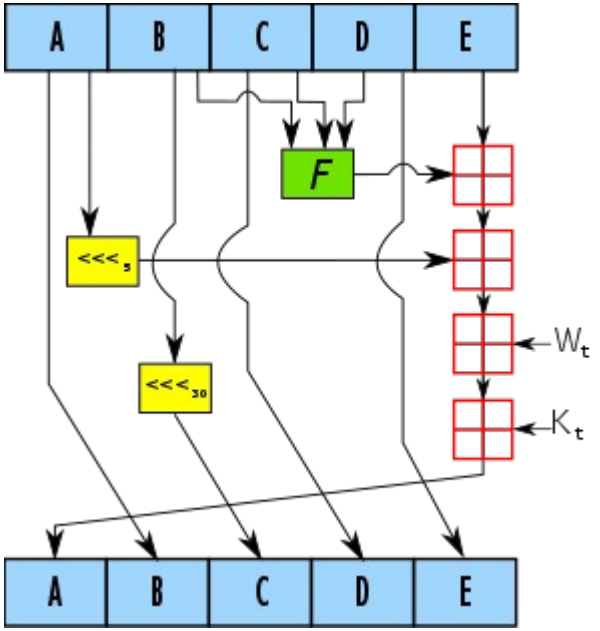
Liste des fonctions de hachage

Les principales fonctions de hachage sont :

- **MD5**, **maintenant obsolète** et plutôt utilisé pour l'empreinte numérique d'un fichier

Les fonctions **SHA** ont été publiées par la NSA (*National Security Agency*)

- SHA-1 (dépassée car trop courte)
- SHA-256 (mots de 32 bits)
- SHA-512 (mots de 64 bits)



Limite des fonctions de hachage

Tout principe de sécurité comporte intrinsèquement une possibilité de faille car Ne pas trouver le hash d'un mot de passe n'empêche **pas les attaques :**

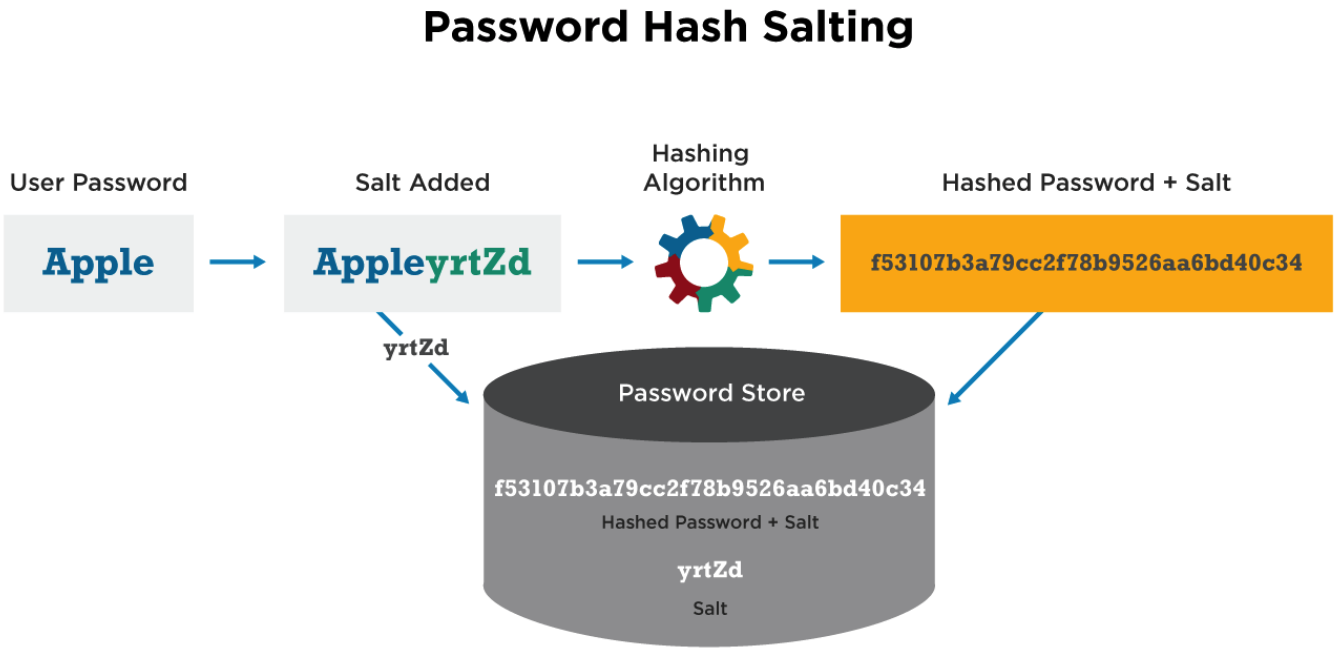
- **« force brute »,**
- **analyse séquentielle,**
- **« rainbow tables »**
- **ou simplement dictionnaire**

Ces attaques sont rendues plus faciles maintenant avec des algorithmes plus performants et des puissances de calcul plus importantes

Principe du « salage »

Pour essayer d'y répondre, on **ajoute un SALT** (« salage » en français) :

- On ajoute **une donnée supplémentaire (concaténation)**, souvent aléatoire et dynamique
- Le salt est lui aussi **stocké dans la base de données**



Principe en PHP (1)

ETAPE 1 : Création du SALT (ici cryptographe PRNG)

```
byte[] salt;
new RNGCryptoServiceProvider().GetBytes(salt = new byte[16]);
```

ETAPE 2 : Création de Rfc2898DeriveBytes et obtention de la valeur de hachage

```
var pbkdf2 = new Rfc2898DeriveBytes(password, salt, 100000);
byte[] hash = pbkdf2.GetBytes(20);
```

ETAPE 3 : Combinaison du SALT et du mot de passe

```
byte[] hashBytes = new byte[36];
Array.Copy(salt, 0, hashBytes, 0, 16);
Array.Copy(hash, 0, hashBytes, 16, 20);
```

ETAPE 4 : Traduire la combianison SALT + hash dans une chaîne pour le stockage

```
string savedPasswordHash = Convert.ToBase64String(hashBytes);
DBContext.AddUser(new User { ..., Password = savedPasswordHash });
```

Principe en PHP (2)

ETAPE 5 : Vérification d'un mot de passe utilisateur

```
/* Fetch the stored value */
string savedPasswordHash = DBContext.GetUser(u => u.UserName == user).Password;
/* Extract the bytes */
byte[] hashBytes = Convert.FromBase64String(savedPasswordHash);
/* Get the salt */
byte[] salt = new byte[16];
Array.Copy(hashBytes, 0, salt, 0, 16);
/* Compute the hash on the password the user entered */
var pbkdf2 = new Rfc2898DeriveBytes(password, salt, 100000);
byte[] hash = pbkdf2.GetBytes(20);
/* Compare the results */
for (int i=0; i < 20; i++)
    if (hashBytes[i+16] != hash[i])
        throw new UnauthorizedAccessException();
```