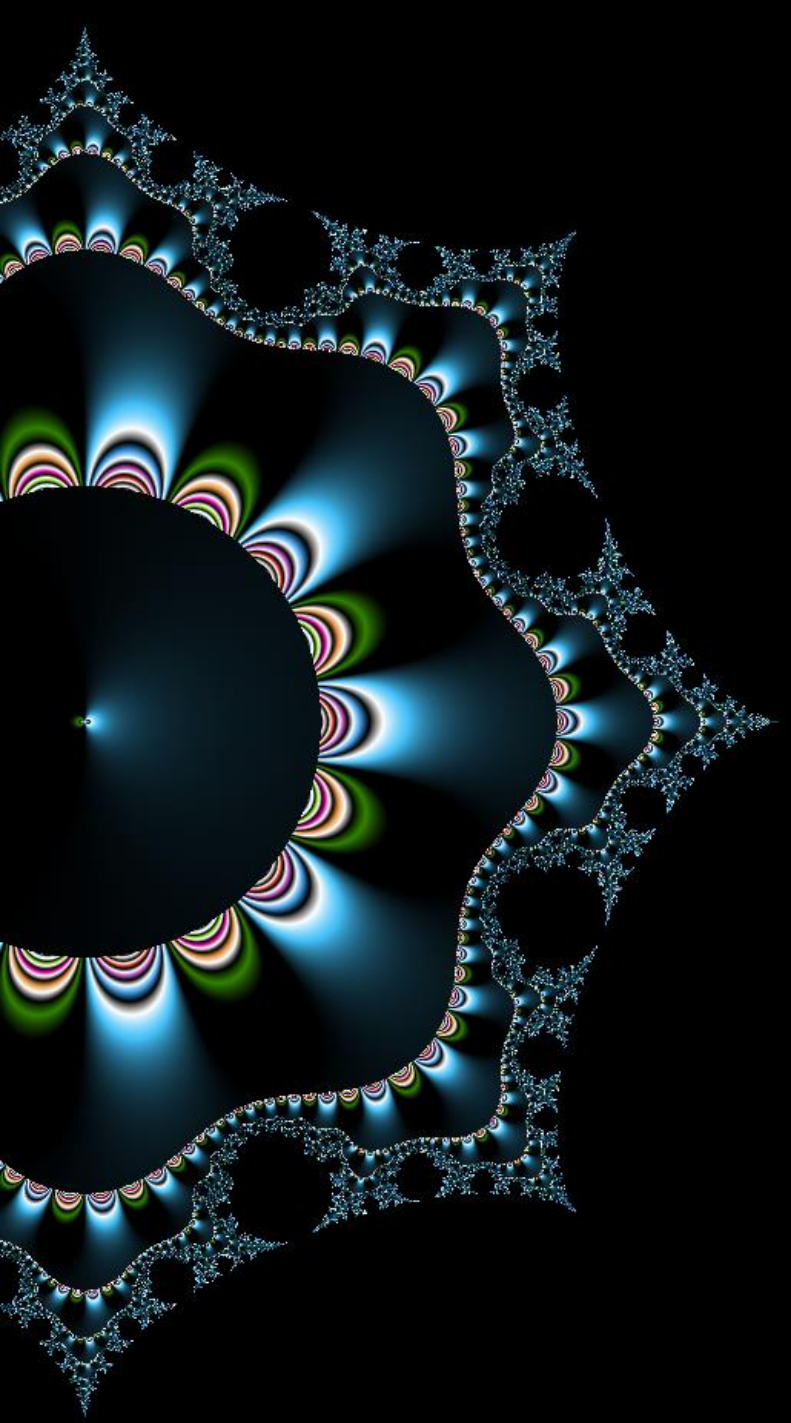


06 Cryptographie

Les mots de passes

Corinne HENIN

www.arsouyes.org

Quel est le problème

Accès frauduleux à la BDD

Aucun système n’est sûr

Mauvaise conf, vulnérabilité, insider, …

Défense en profondeur

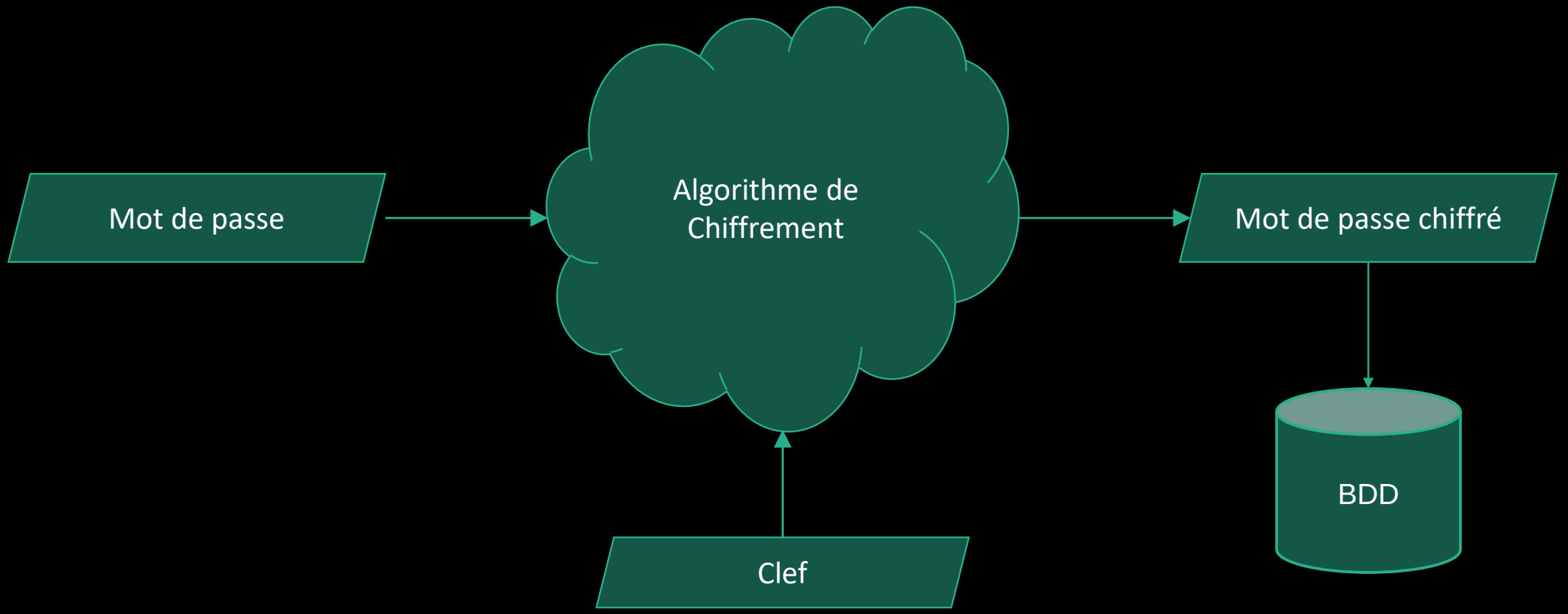
Défendre à tous les niveaux

Protéger le contenu de la BDD

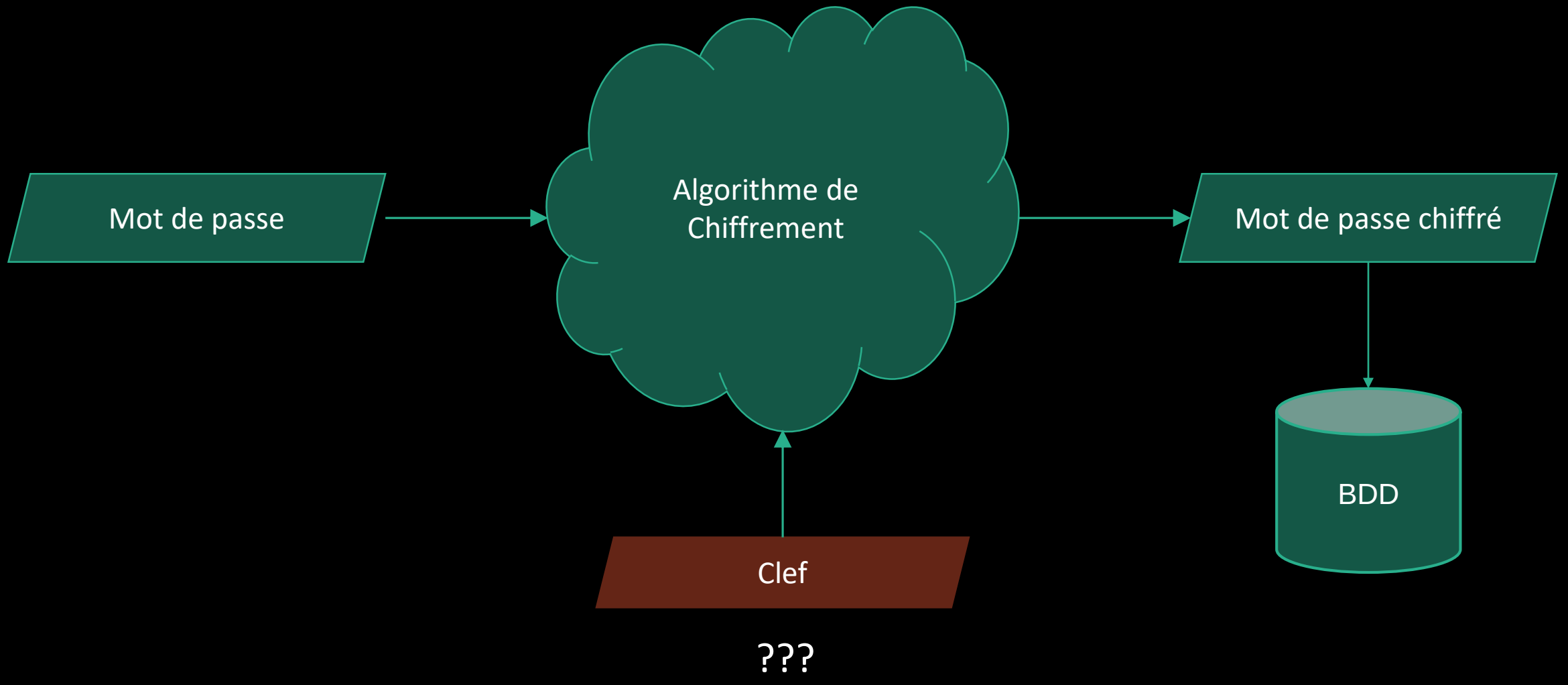
Intuitivement

Solution un peu lourde

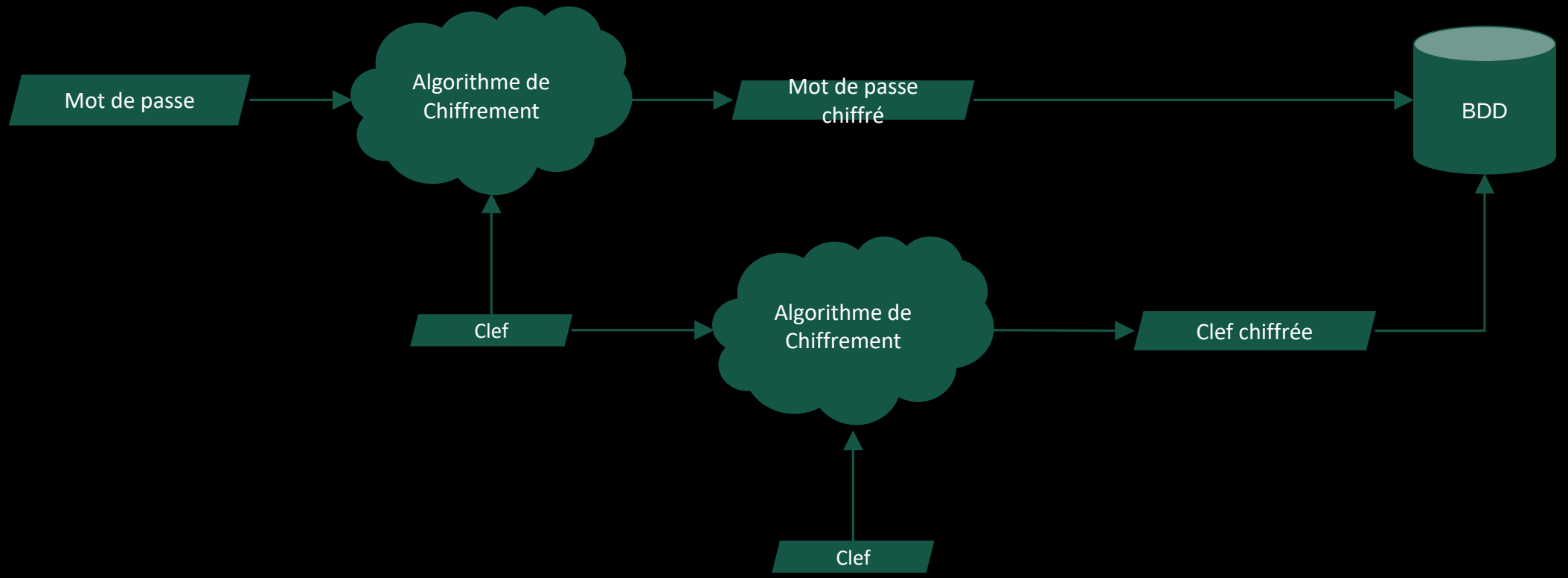
Chiffrer


Mot de passe
Algorithme de Chiffrement
Mot de passe chiffré
BDD
Clef

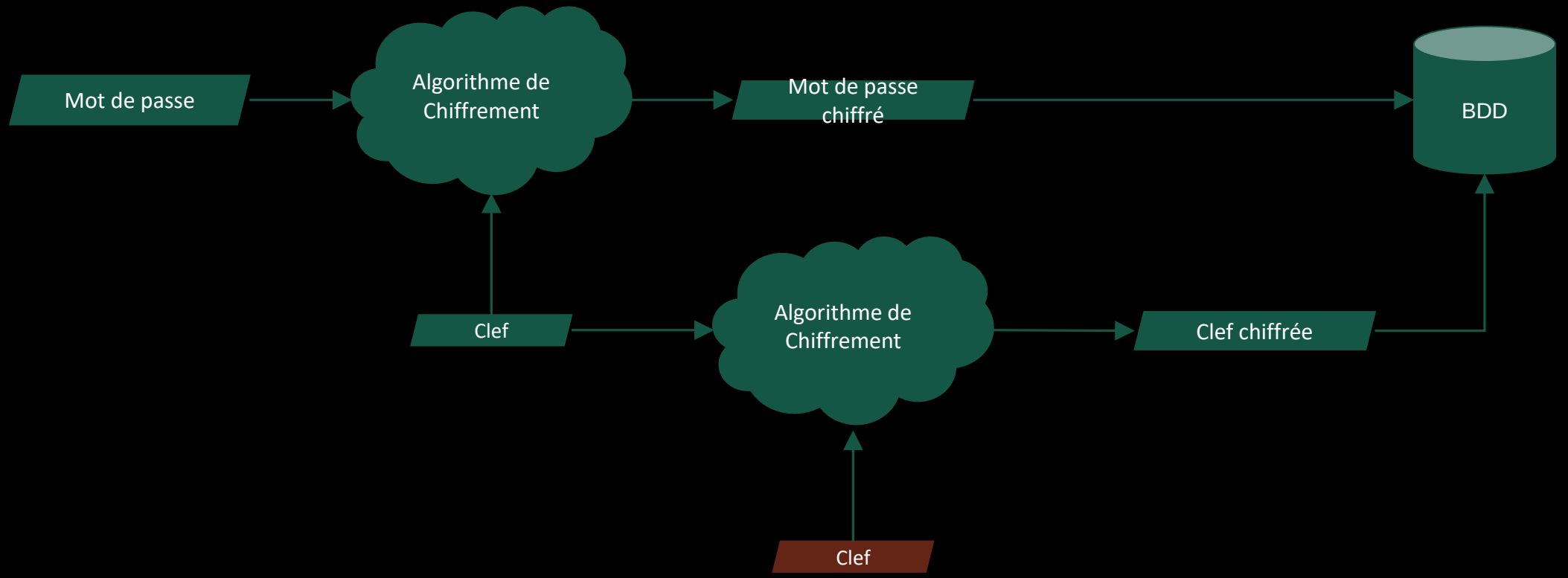
Chiffrer


Mot de passe
Algorithme de Chiffrement
Mot de passe chiffré
BDD
Clef
???

Chiffrer


Mot de passe
Algorithme de Chiffrement
Mot de passe chiffré
BDD
Clef
Algorithme de Chiffrement
Clef chiffrée
Clef

Chiffrer

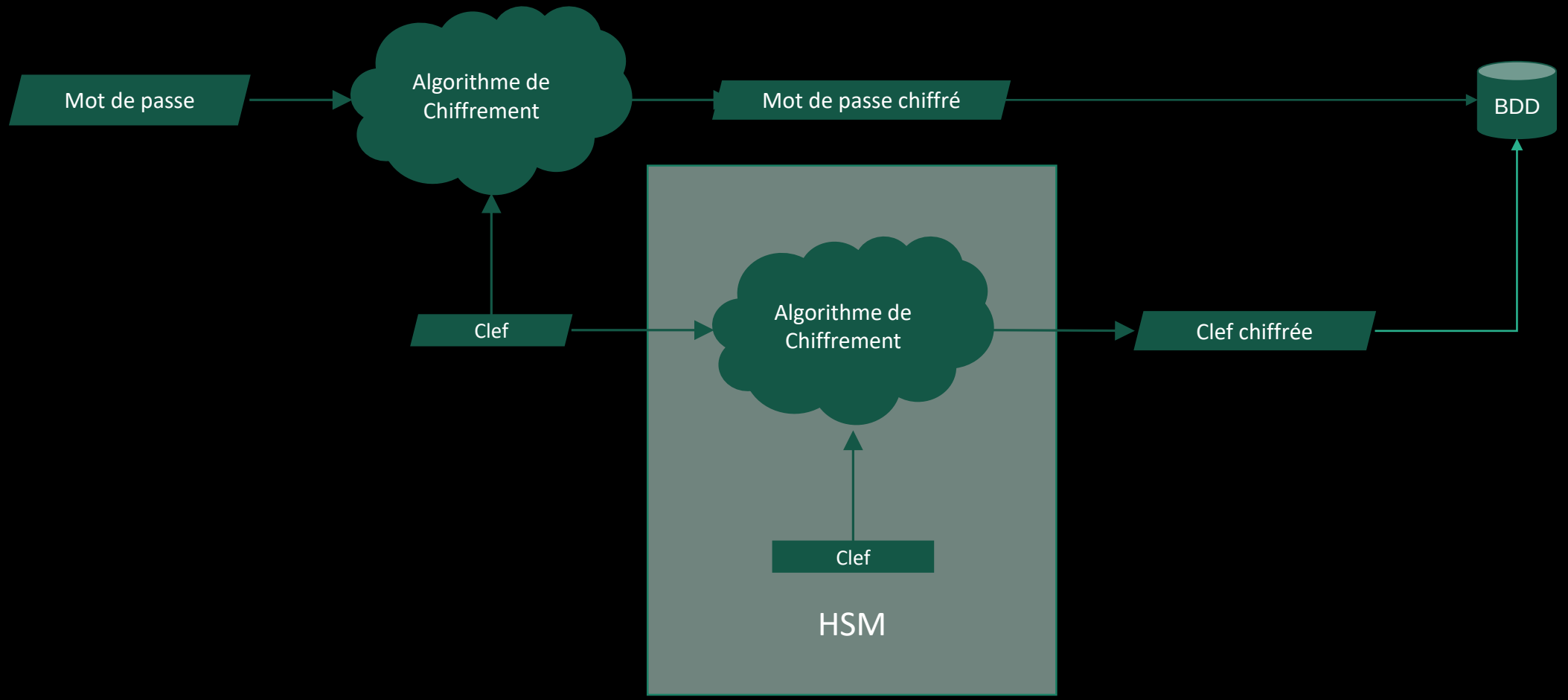

Mot de passe
Algorithme de Chiffrement
Mot de passe chiffré
BDD
Clef
Algorithme de Chiffrement
Clef chiffrée
Clef

Chiffrer


Mot de passe
Algorithme de Chiffrement
Mot de passe chiffré
BDD
Clef
Algorithme de Chiffrement
Clef chiffrée
Clef
Algorithme de Chiffrement
Clef chiffrée
Clef
Algorithme de Chiffrement
Clef chiffrée
Clef
Algorithme de Chiffrement
Clef chiffrée
Clef

.....

Chiffrer


Mot de passe
Algorithme de Chiffrement
Mot de passe chiffré
BDD
Clef
Algorithme de Chiffrement
Clef chiffrée
Clef
HSM

Limitations

C’est lourd

Code plus volumineux, achat de matériel

C’est complexe

Changement de clefs

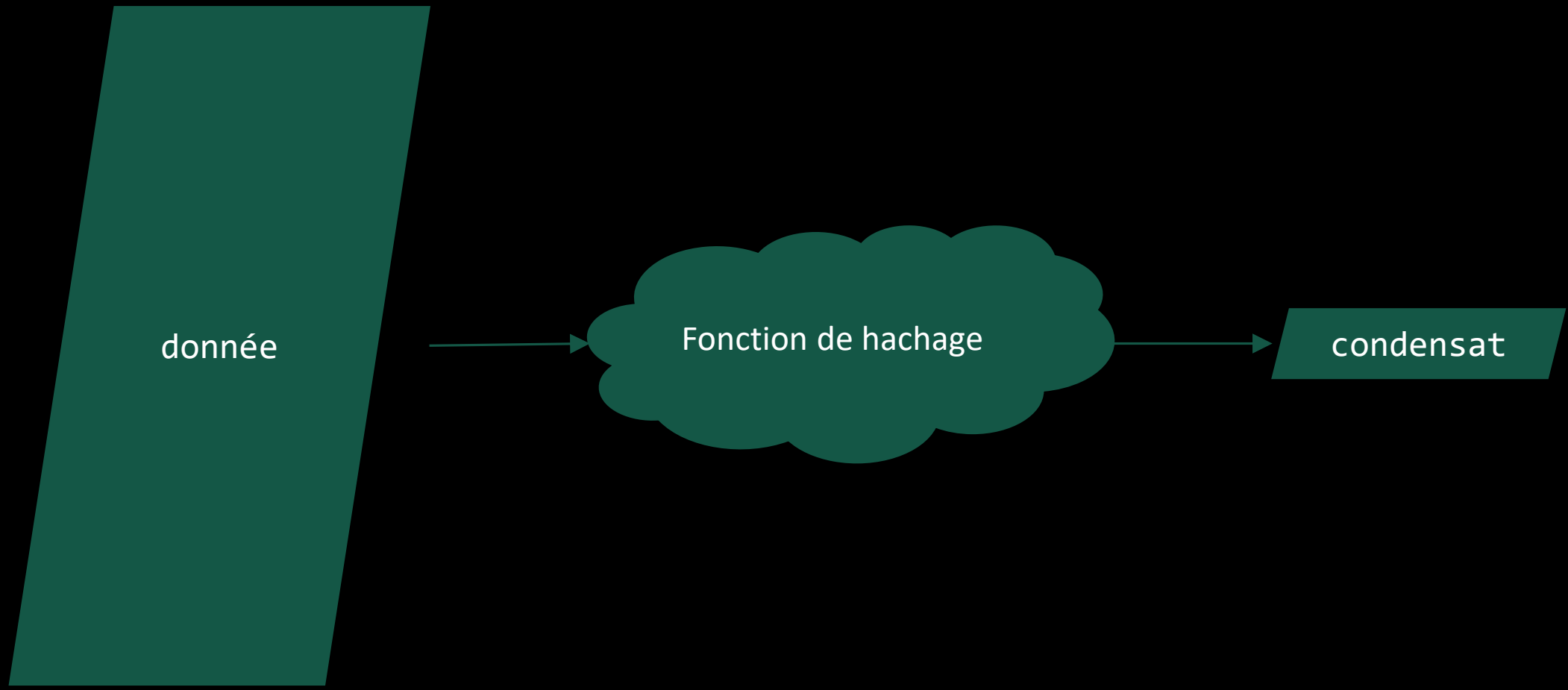
Sauvegarde obligatoire

Destruction des données si ouverture

Penser autrement

Autre solution ?!

Fonction de hachage



Critères de sécurité

Irréversible

ne pas trouver la donnée

Infalsifiable

ne pas trouver une autre donnée

Résistante aux collisions

Sert aux preuves formelles

Exemples de fonctions publiées

Obsolètes

MD5

(depuis 1993)

SHA1

(depuis 2012)

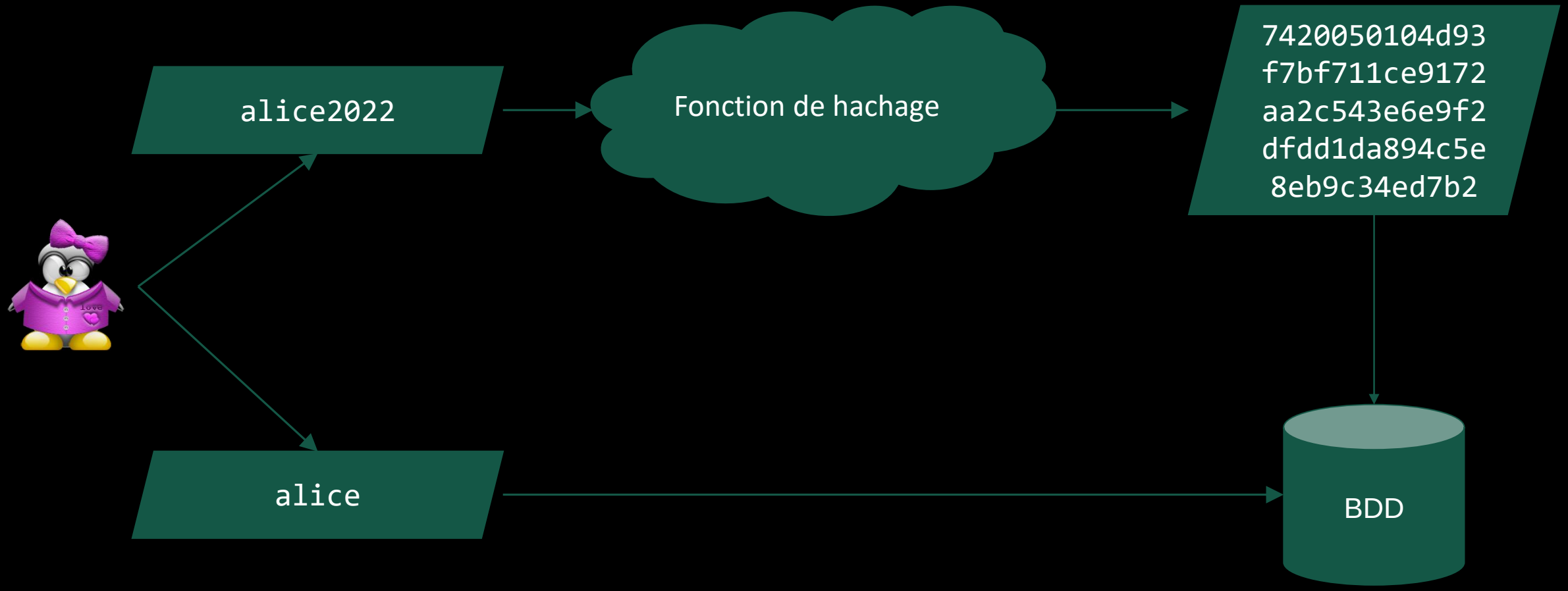
A jour

SHA2

(SHA256 et SHA512)

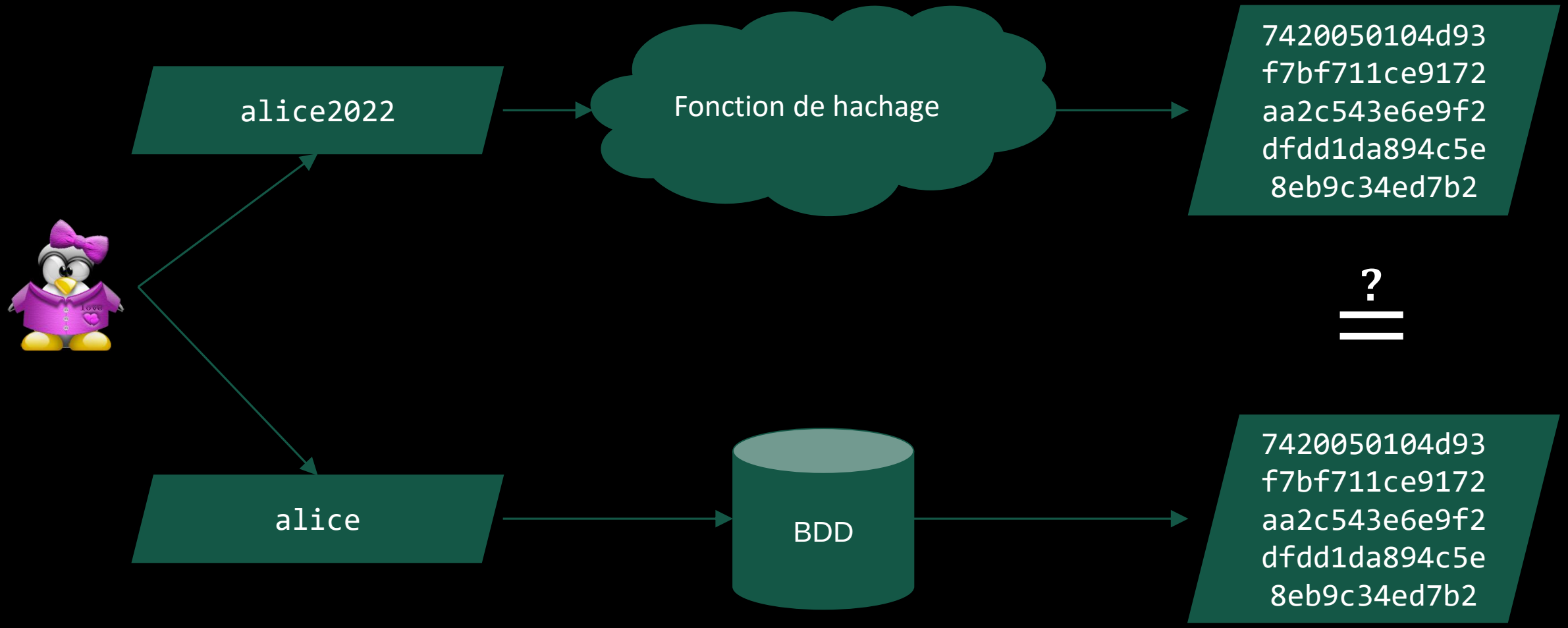
Nouvel utilisateur

Stockage du mot de passe



Authentification

Vérification du mot de passe



Php – Stockage

```
function store($password) {
    return hash("sha512", $password) ;
}

function verify($password, $stored) {
    return hash_equals(store($password), $stored) ;
}
```

Php – Vérification

```
function store($password) {
    return hash("sha512", $password) ;
}

function verify($password, $stored) {
    return hash_equals(store($password), $stored) ;
}
```

Limitations

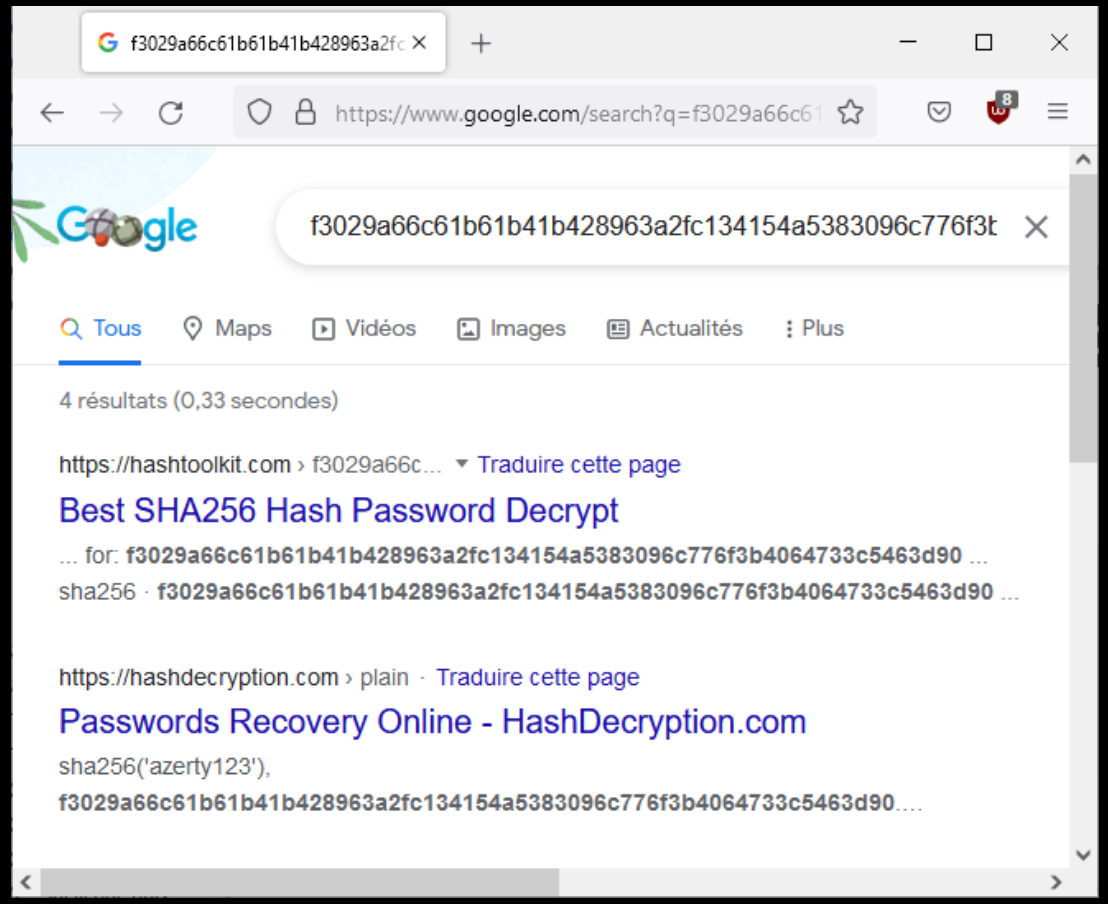
Hachage déterministe

Dictionnaire

(pré-calcul)

Rainbowtable

(dictionnaire compressé)



Ajouter du sel

Pour ajouter du piment

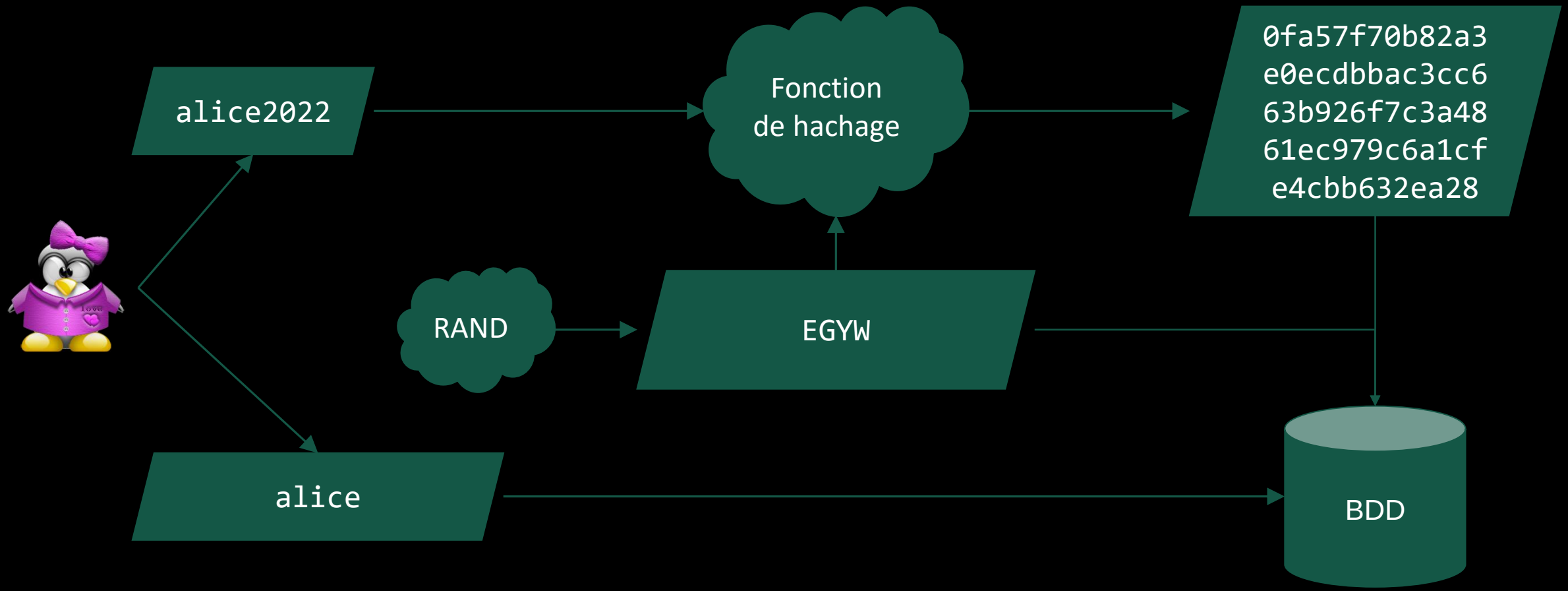
Salage

Augmenter artificiellement la taille

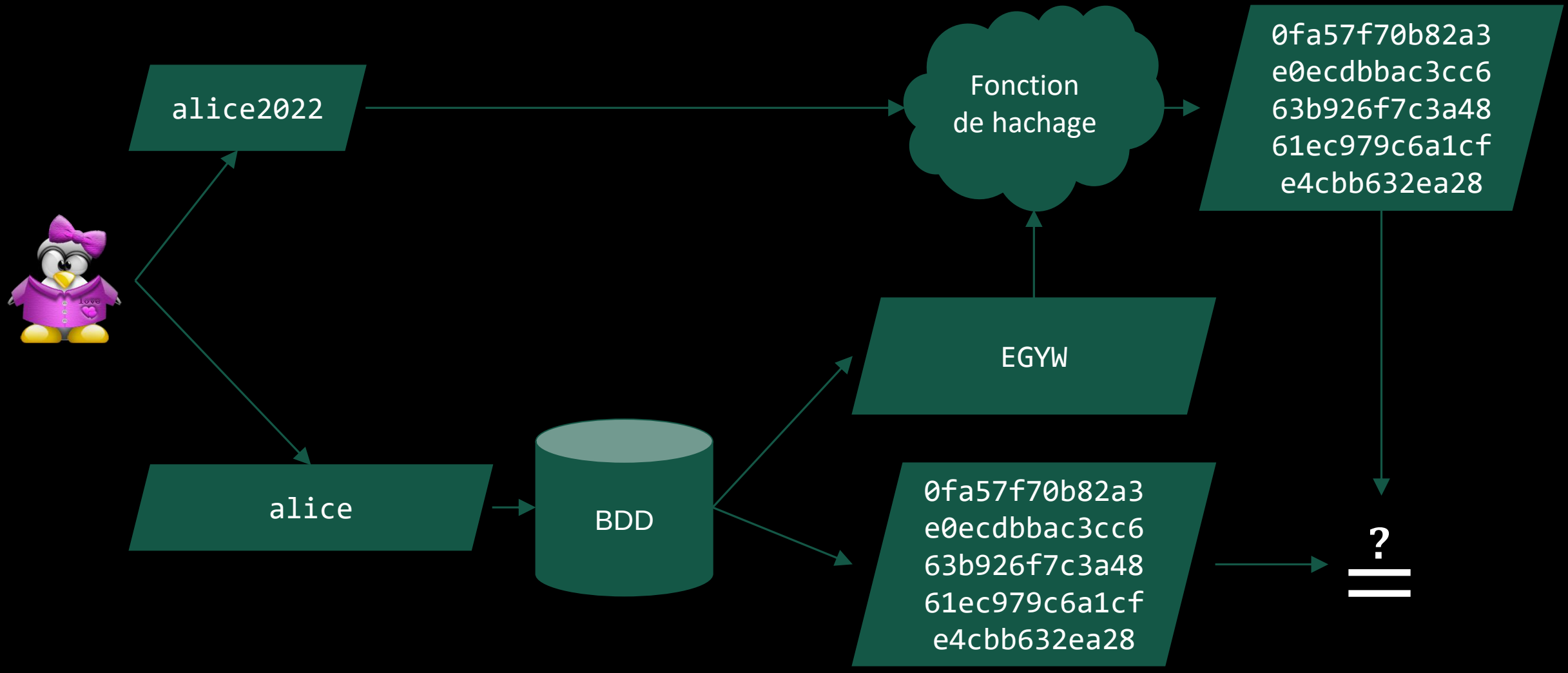
Ajout de caractères aléatoires

Sel stocké dans la BDD

Stockage du mot de passe



Vérification d’un mot de passe



Php – Stockage

```php
function store($password) {
    $salt = bin2hex(random_bytes(16)) ; // 128 bits
    $hash = hash("sha512", $password . $salt) ;
    return "$salt.$hash" ;
}

function verify($password, $stored) {
    list($salt, $hash) = explode(".", $stored) ;
    return hash_equals(hash("sha512", $password . $salt), $hash) ;
}
```

Php – Vérification

```
function store($password) {
    $salt = bin2hex(random_bytes(16)) ; // 128 bits
    $hash = hash("sha512", $password . $salt) ;
    return "$salt.$hash" ;
}

function verify($password, $stored) {
    list($salt, $hash) = explode(".", $stored) ;
    return hash_equals(hash("sha512", $password . $salt), $hash) ;
}
```

Avantage

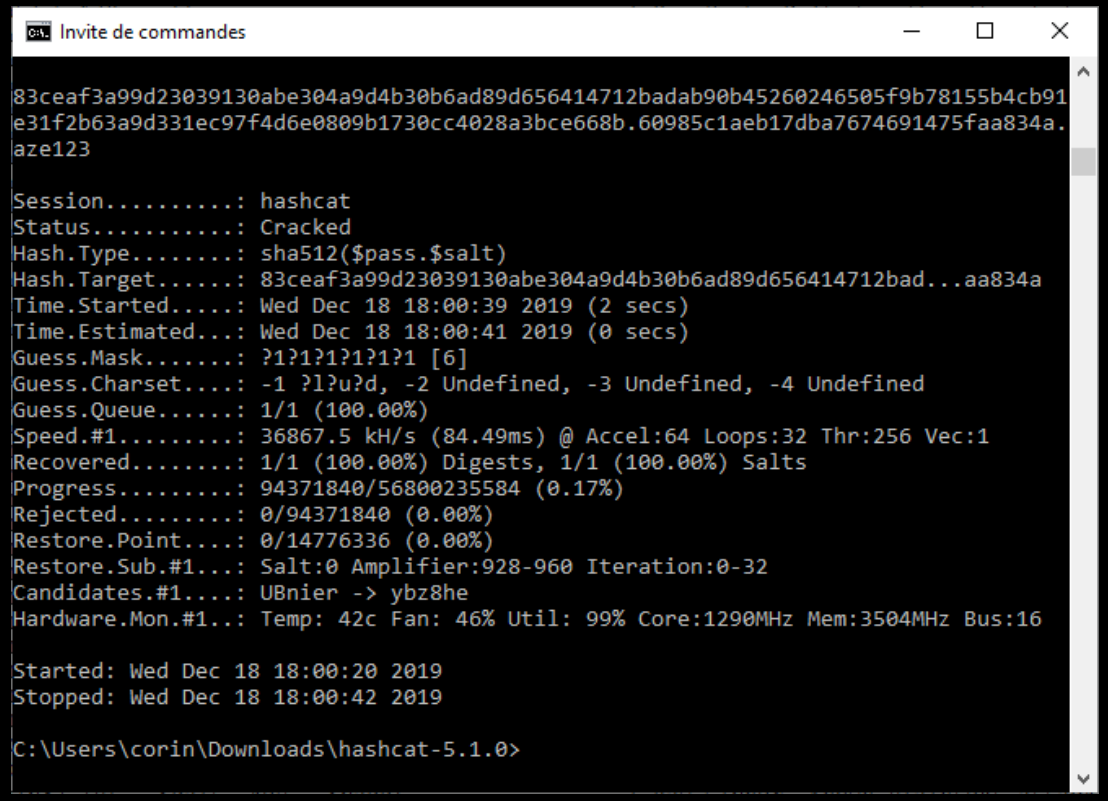
Bruteforce nécessaire

(dictionnaires plus stockables)

Limitations

GPU efficaces

(i.e. Hashcat efficace)



Ralentir

Pour ne plus afficher trop vite

Principe

Augmenter le temps

Répétition des opérations

Augmenter les ressources

Vecteur d'initialisation _très_ grand

Parallélisme

Fonctions adaptatives

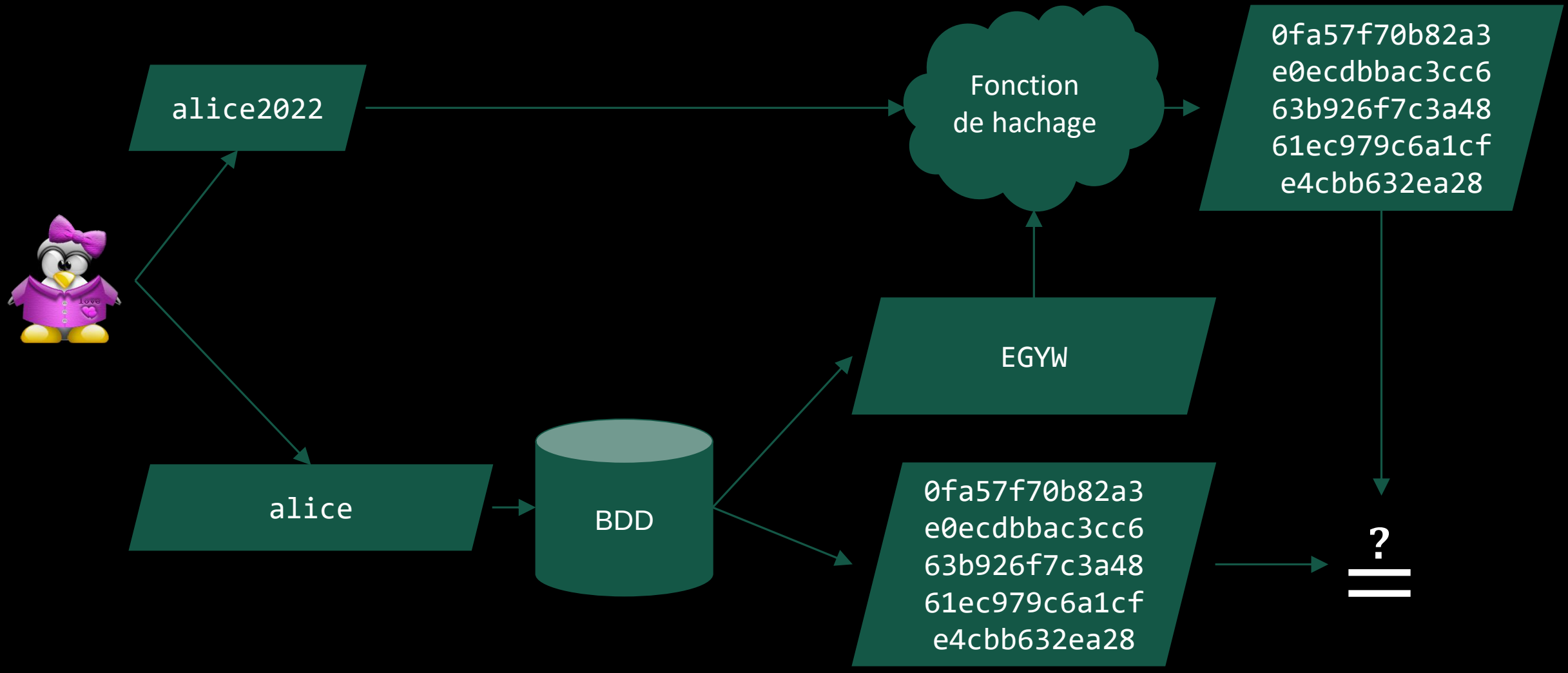
Bcrypt (1999)

Scrypt (2012)

Argon2 (2015)

PBKDF2 (2015)

Vérification d’un mot de passe



Php – Hachage

```
function myHash($password, $salt, $cost) {
    $options = sprintf('$2a$%\'.02d$%\'.22s$', $cost, $salt) ;
    return crypt($password, $options) ;
}

function store($password) {
    $salt = bin2hex(random_bytes(16)) ;
    $hash = myHash($password, $salt, 10) ;
    return $hash ;
}

function verify($password, $stored) {
    $hash = crypt($password, $stored) ;
    return hash_equals($stored, $hash) ;
}
```

Php – Stockage

```
function myHash($password, $salt, $cost) {
    $options = sprintf('$2a$%\'.02d$%\'.22s$', $cost, $salt) ;
    return crypt($password, $options) ;
}

function store($password) {
    $salt = bin2hex(random_bytes(16)) ;
    $hash = myHash($password, $salt, 10) ;
    return $hash ;
}

function verify($password, $stored) {
    $hash = crypt($password, $stored) ;
    return hash_equals($stored, $hash) ;
}
```

Php – Vérification

```
function myHash($password, $salt, $cost) {
    $options = sprintf('$2a$%\'.02d$%\'.22s$', $cost, $salt) ;
    return crypt($password, $options) ;
}

function store($password) {
    $salt = bin2hex(random_bytes(16)) ;
    $hash = myHash($password, $salt, 10) ;
    return $hash ;
}

function verify($password, $stored) {
    $hash = crypt($password, $stored) ;
    return hash_equals($stored, $hash) ;
}
```

Ne pas réinventer la roue

D’autres y on déjà pensé pour vous

Php – Stockage

```
function store($password) {
    return password_hash($password, PASSWORD_DEFAULT) ;
}

function verify($password, $stored) {
    return password_verify($password, $stored) ;
}
```

Php – Stockage

```
function store($password) {
    return password_hash($password, PASSWORD_DEFAULT) ;
}

function verify($password, $stored) {
    return password_verify($password, $stored) ;
}
```