# Algorithmes de hachage

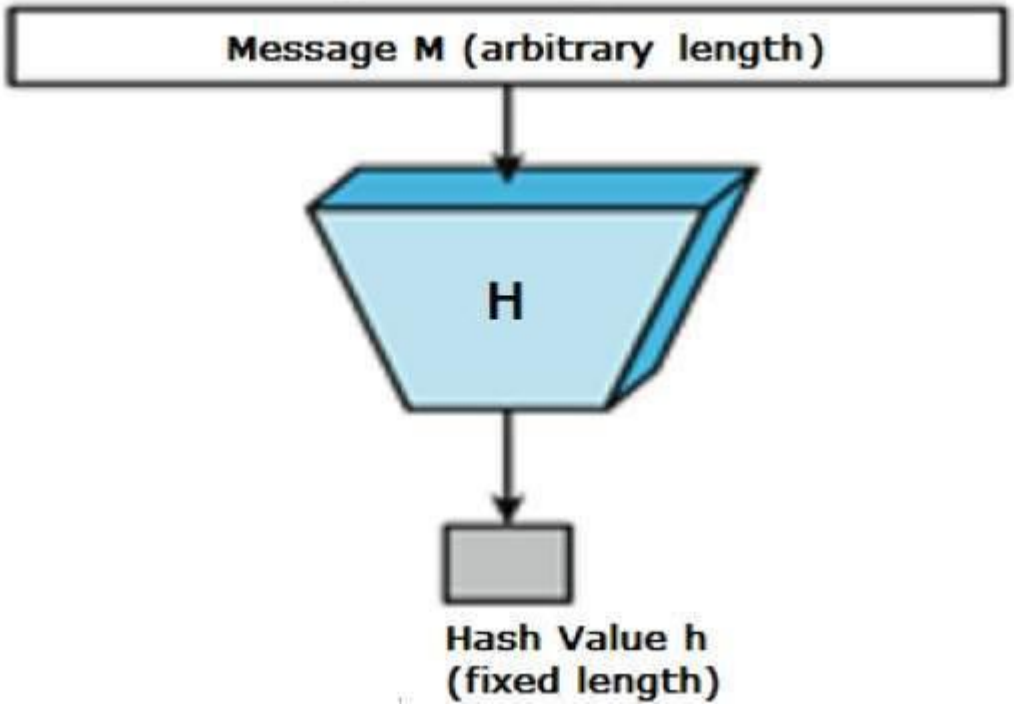
# Fonctions de Hachage

# fonction de hachage

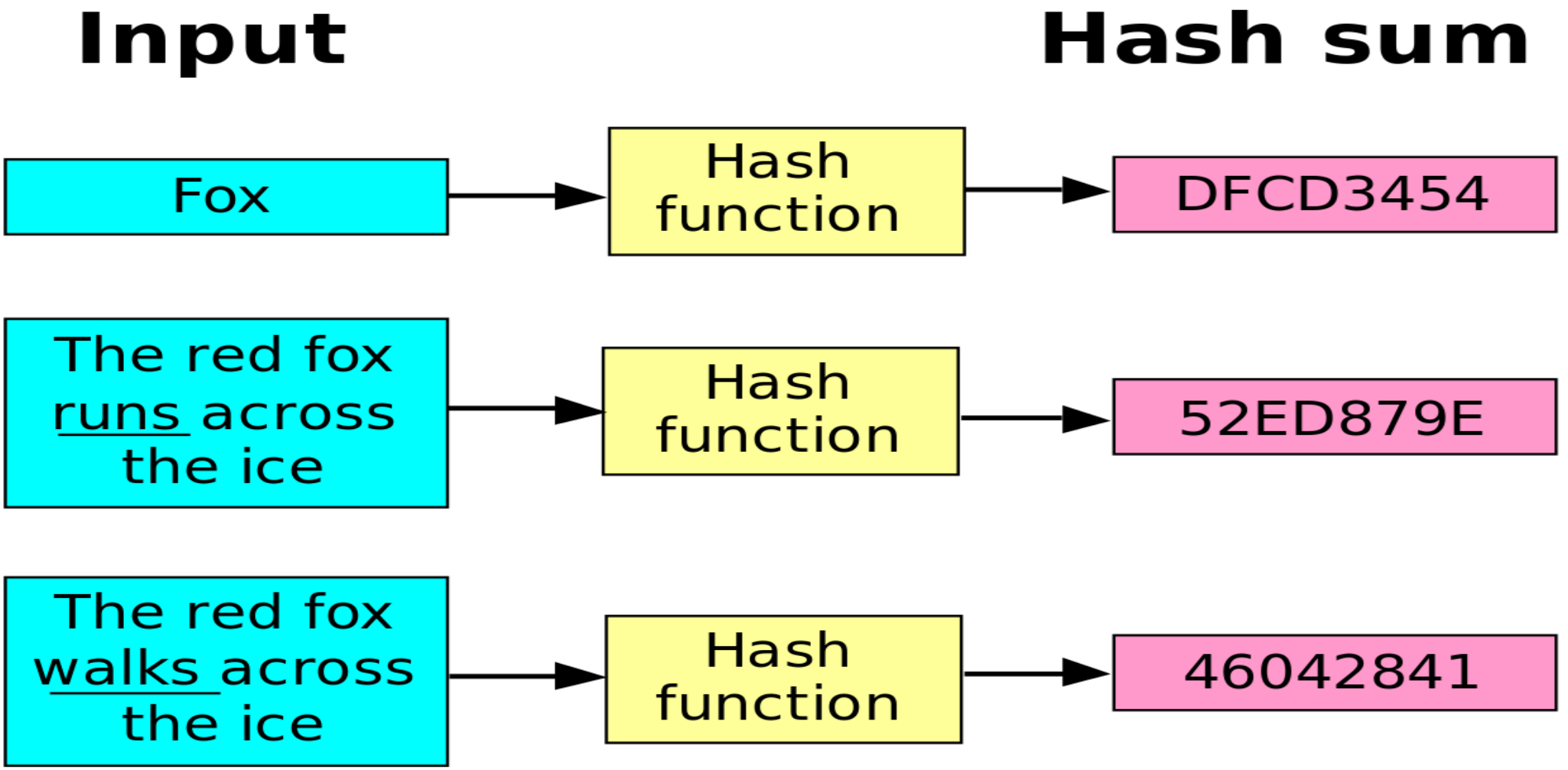
.**fonction de hachage** (hash : hash function) , une fonction particulière qui, à partir d'une donnée fournie en entrée, calcule une empreinte servant à identifier rapidement la donnée initiale, au même titre qu'une signature pour identifier une personne.

- Les fonctions de hachage sont utilisées en informatique et en cryptographie notamment pour reconnaître rapidement des fichiers ou des mots de passe.

# fonction de hachage



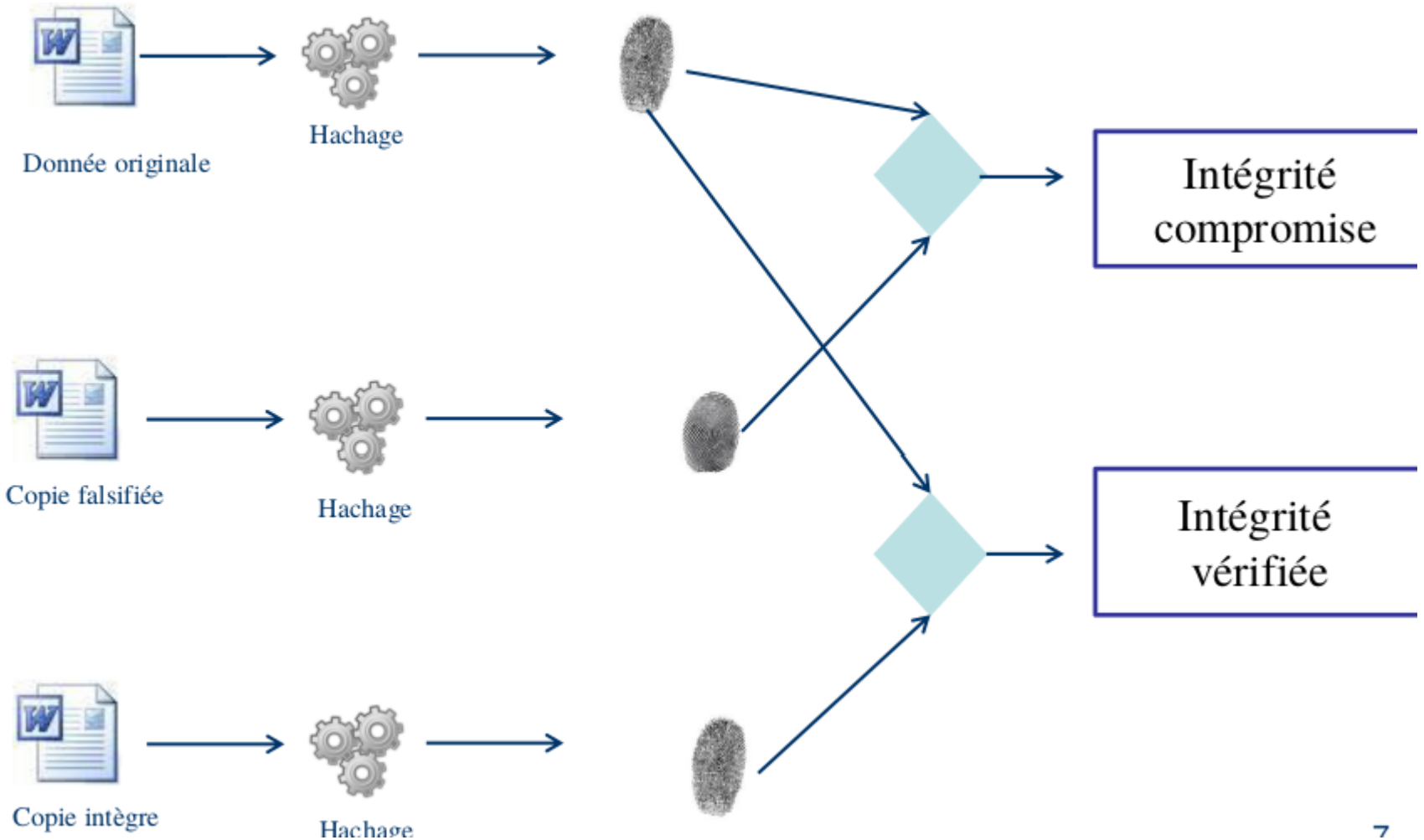
# fonction de hachage



# Résistance à la préimage

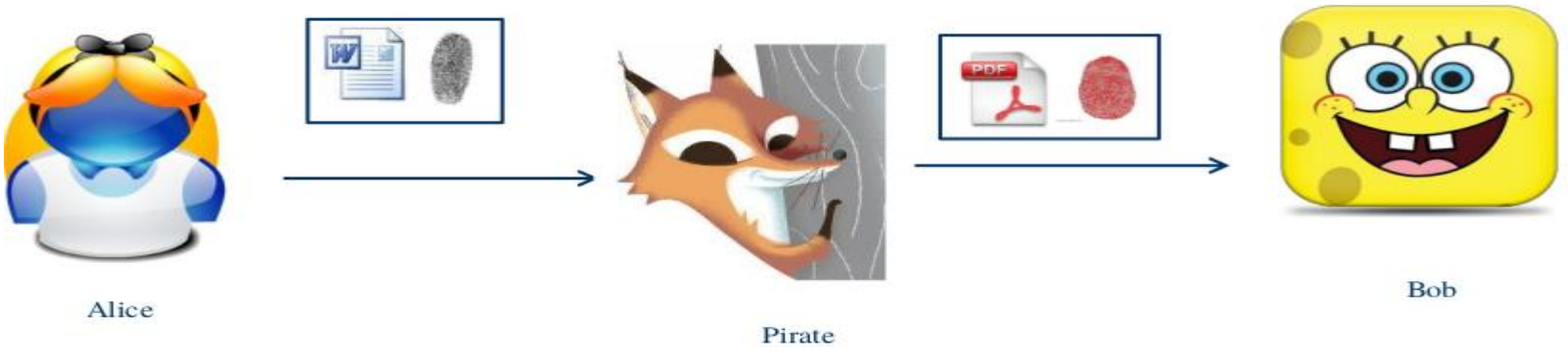
- Une fonction h est résistante à la préimage s'il est difficile de trouver à partir d'une empreinte donnée p un message m tel que h(m) = p.
- Une fonction h est résistante à la seconde préimage s'il est difficile de trouver à partir d'un message m donné un autre message m' tel que h(m) = h(m')
- Une fonction de hachage h est résistante aux collisions s'il est difficile de trouver deux messages m et m' tel que h(m) = h(m')

# Processus de vérification



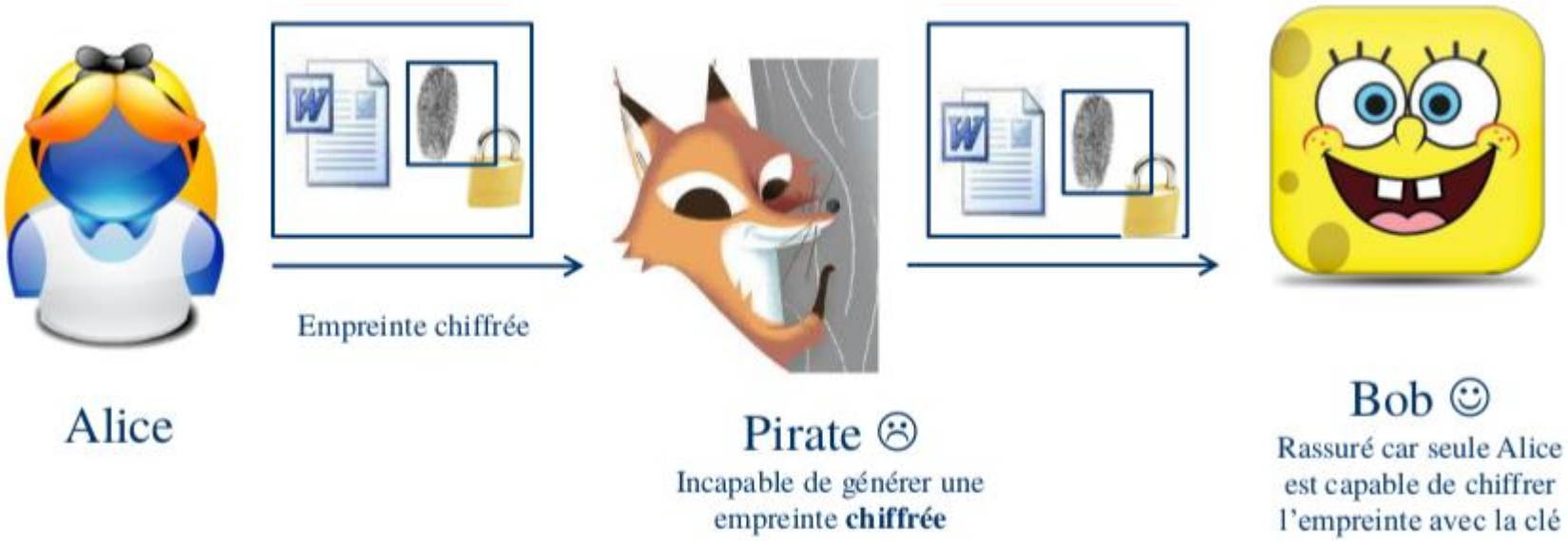
# fonction de hachage

- Très importante dans l’échange de messages
- MITM (Man-In-The-Middle)
- Recevoir une paire <message, empreinte> n’assure aucune garantie sur l’origine du message



# fonction de hachage

- Utiliser une clé secrète



# Utilisation

Pour vérifier l’intégrité d’un message

Si le message et son empreinte ne correspondent plus, il y a eu une manipulation du message.

Pour la signature numérique

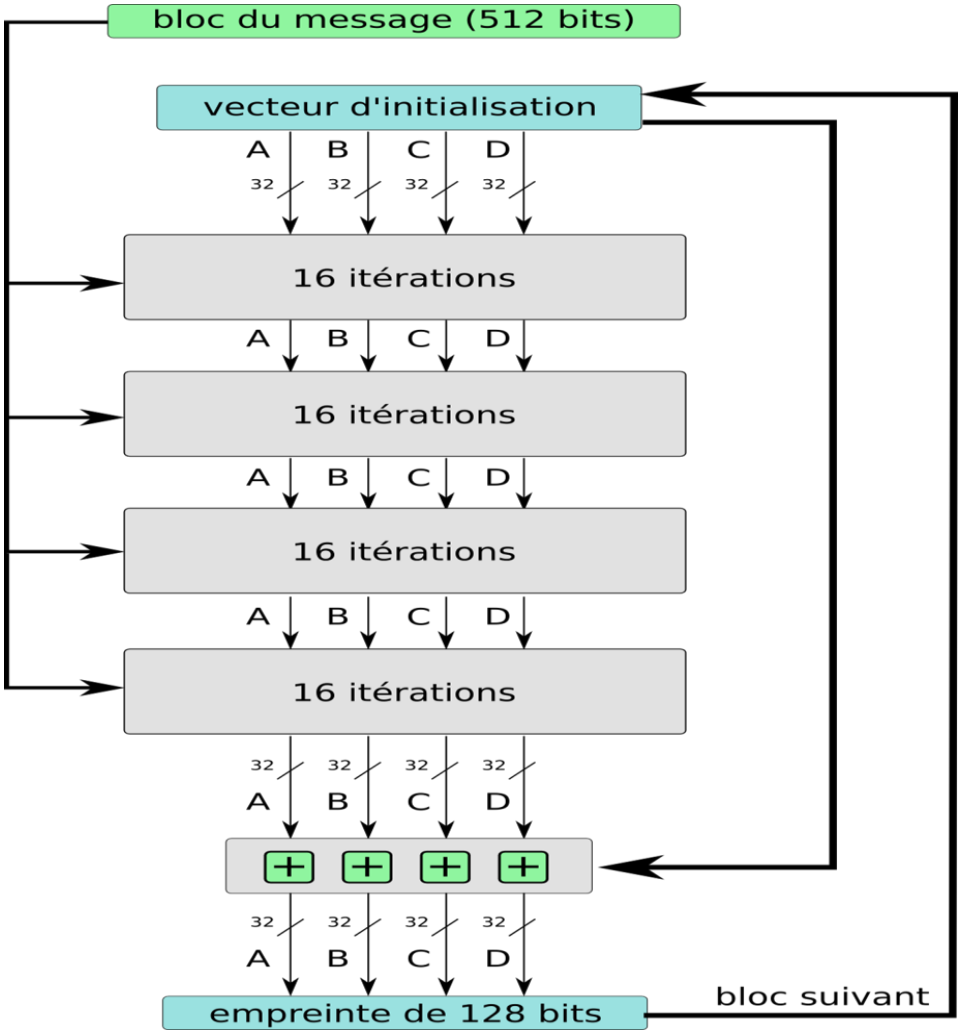
Afin de permettre de signer un message de longueur arbitrairement long,l’empreinte du message est signée au lieu du message proprement dit.

# MD5

MD2, MD4 et MD5

- Message Digest: 1989, 1990 et 1991 par Ronald Rivest (MIT)
- Empreintes de 128 bits.
- Vulnérables aux collisions: messages différents avec une même empreinte.

# MD5


bloc du message (512 bits)
vecteur d'initialisation
A B C D
32 32 32 32
16 itérations
A B C D
16 itérations
A B C D
16 itérations
A B C D
16 itérations
32 32 32 32
A B C D
32 32 32 32
A B C D
empreinte de 128 bits
bloc suivant

# SHA

- SHA
- – Secure Hashing Algorithm
- – National Institute of Standards and Technology (NIST)
- SHA-1 (160 bits)
- – Vulnérable aux collisions.
- – Publiée en 1995. Des vulnérabilités publiées en 2005.

# SHA

- SHA- 2
- 4 versions: 224, 256, 384 et 512 bits
- Publiée en 2001
- Non affectée par les vulnérabilités SHA-1!
- SHA-256 vulnerable aux collisions

# Table de hachage

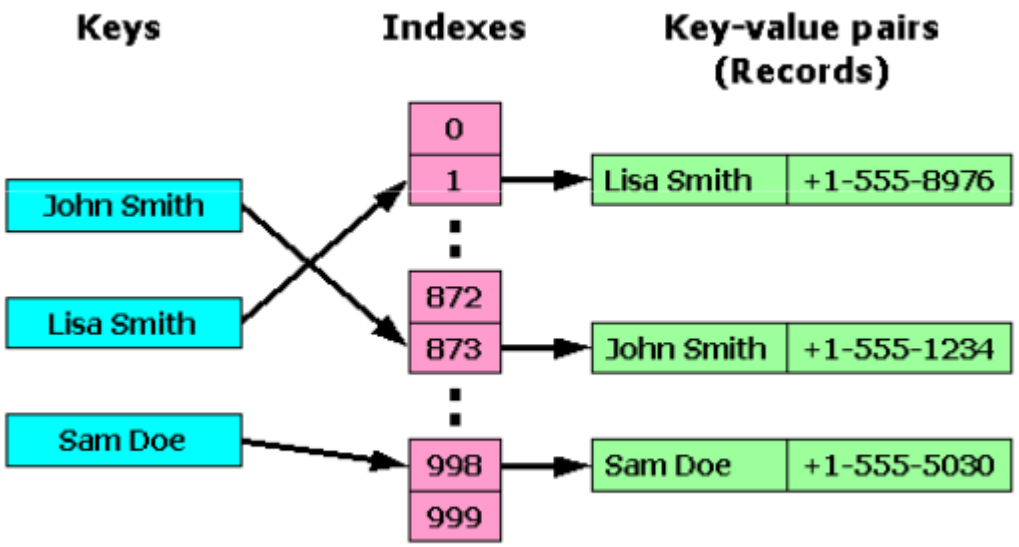
# Table de hachage

- Une  table hachage(**hash table**) est une structure de données qui permet une association clé-élément
- On accède à chaque élément de la table via sa clé.
- L'accès à un élément se fait en transformant la clé en une valeur de hachage (ou simplement hachage) par l'intermédiaire d'une fonction de hachage.
- Le hachage est un nombre qui permet la localisation des éléments dans le tableau, typiquement le hachage est l'index de l'élément dans le tableau.
- Une case dans le tableau est appelée alvéole**(bucket/slot).**

# Table de hachage

- **insérer**(H,elt,clé) : insère dans la table de hachage H un élément elt dont la clef est clé
- elt **recherche**(H,clé) : recherche dans la table de hachage H si un élément est associé à la clef clé et renvoie cet élément

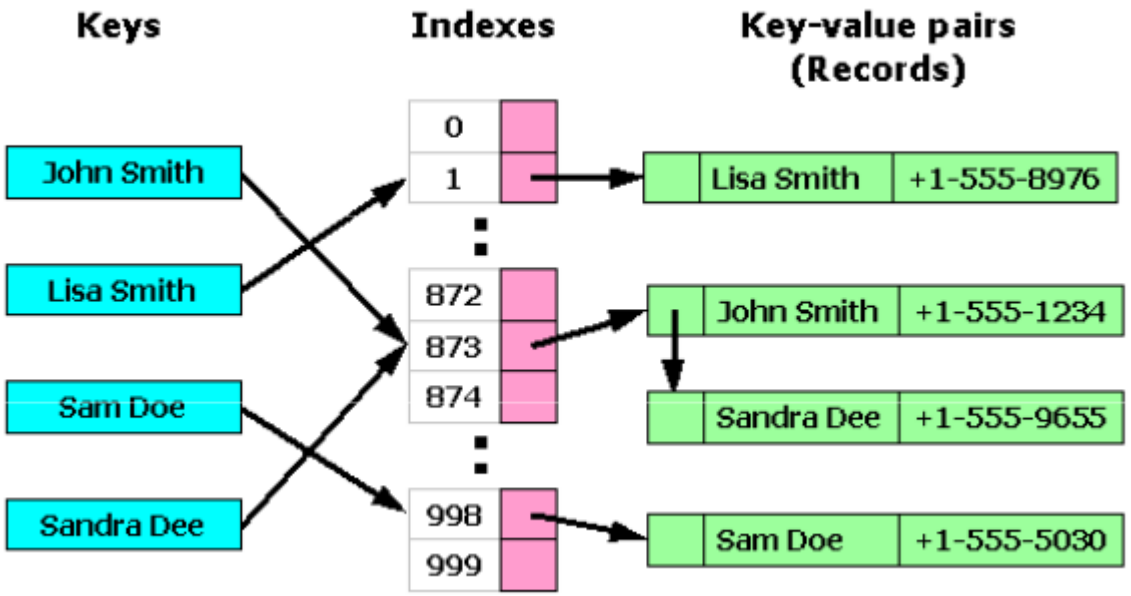
# Table de hachage



# Table de hachage

- Les tables de hachage permettent
  - un accès en O(1) en moyenne, mais
  - le temps d'accès dans  le pire des cas peut être de O(n).
- Une table de hachage n’est pas ordonnée
- Le fait de créer un hash à partir d'une clé peut engendrer un problème de collision
  - accès à la même a position dans le "tableau"
- Pour minimiser les risques de collisions, il faut donc choisir soigneusement sa fonction de hachage.

# Collision


Keys
Indexes
Key-value pairs (Records)
John Smith
Lisa Smith
Sam Doe
Sandra Dee
0
1
872
873
874
998
999
Lisa Smith
+1-555-8976
John Smith
+1-555-1234
Sandra Dee
+1-555-9655
Sam Doe
+1-555-5030

# Résolution des collisions

- Les Collisions sont fréquentes
- De nombreuses stratégies de résolution des collisions existent mais les plus connues et utilisées sont

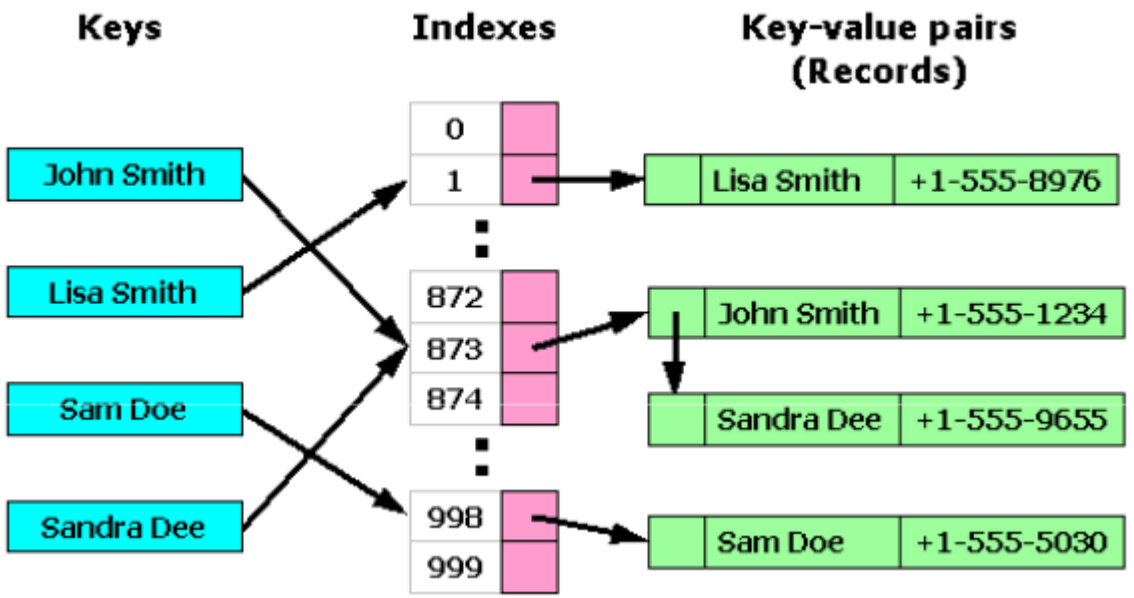
  -le chainage

  -l'adressage ouvert.

# Chainage

- Méthode la plus simple.
- Chaque case de la table est en fait une liste chaînée d’éléments dont les clés ont le même hachage.

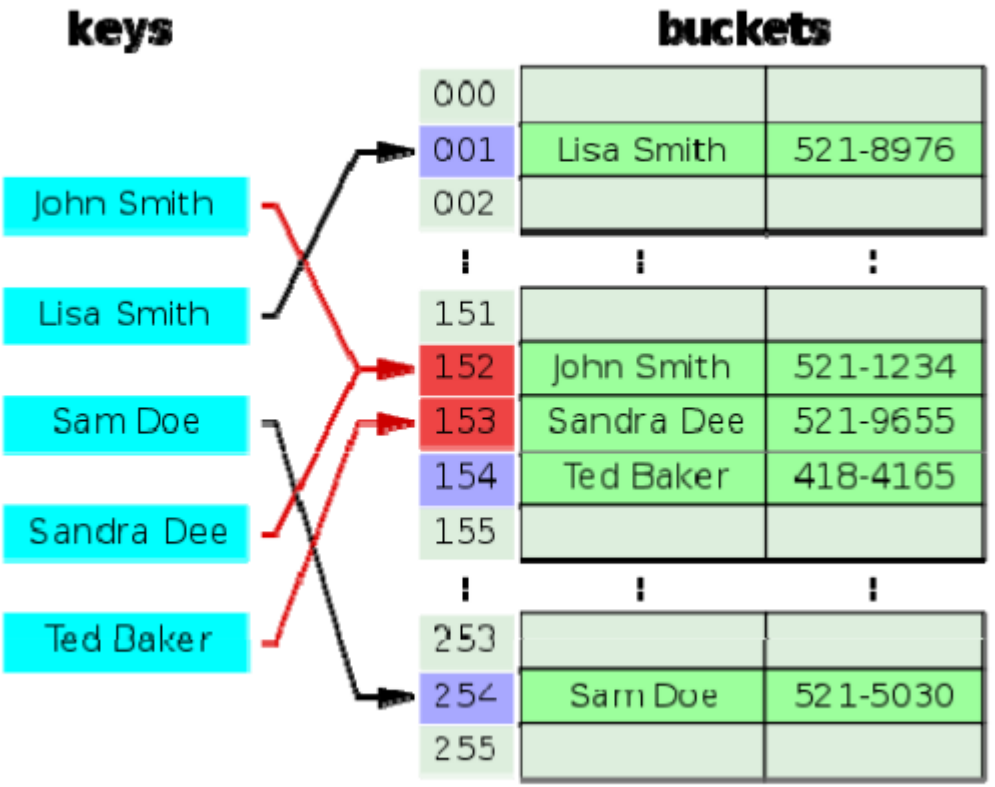
# Chainage


Keys
Indexes
Key-value pairs
(Records)
John Smith
Lisa Smith
Sam Doe
Sandra Dee
0
1
872
873
874
998
999
Lisa Smith
+1-555-8976
John Smith
+1-555-1234
Sandra Dee
+1-555-9655
Sam Doe
+1-555-5030

# Adressage Ouvert

- Adressage ouvert:L'adressage ouvert consiste, dans le cas d'une collision, à stocker les valeurs de hachage dans d'autres alvéoles vides.
- Soit h(k,i) une fonction de sondage. Une méthode de sondage consiste à essayer les alvéoles d'indice h(k,0), h(k,1), ..., jusqu'à ce qu'on trouve une alvéole vide.

# Adressage ouvert



# Adressage Ouvert (Open addressing)

- Sondage linéaire (Linear probing):

$$h(k,i) = (H(k) + i) mod\ N, \quad 0 \leq i < N$$

- Sondage quadratique(Quadratic probing)

$$h(k,i) = (H(k) + c_1 \cdot i + c_2 \cdot i^2) mod\ N, c_2 \neq 0$$

- Double hachage

- H1,H2 deux fonctions de hachage. Il faudra faire en sorte que H2 ne soit jamais égal à 0.

$$h(k,i) = (H_1(k) + i \cdot H_2(k)) mod\ N$$

# Adressage Ouvert

- Recherche :

- Lors d'une recherche, si l'alvéole obtenue par hachage direct ne permet pas d'obtenir la bonne clé, une recherche sur les cases obtenues par une méthode de sondage est effectuée jusqu'à trouver la clé,ou tomber sur une alvéole vide, ce qui indique qu'aucune clé de ce type n'appartient à la table.

# Adressage Ouvert

- Suppression

- Dans un adressage ouvert, la suppression d'un élément de la table de hachage est délicate. . La manière la plus simple de s'en sortir est de ne pas vider l'alvéole , mais d'y placer le mot « Supprimé ». On distinguera ainsi les alvéoles vides des alvéoles où un nom a été effacé « lazy deletion »,

# Performance du cache

- Separate Chaining:s
- Les nœuds d'une liste chaînée peuvent être dispersés en mémoire, causant des défauts de cache.
- Open Addressing:
- Meilleure. La sondage s'effectue dans des emplacements mémoire contigus, ce qui est favorable au cache.

# Hash Table

| Language | Hash Table Name(s) | Collision Strategy | Notes |
|---|---|---|---|
| Java | HashMap<br>Hashtable<br>ConcurrentHashMap | Separate Chaining | Uses linked lists, converts to tree when buckets get large (Java 8+) |
| C++ | std::unordered_map<br>std::unordered_set | Separate Chaining | Standard specifies buckets, implementations use linked lists |
| Python | dict<br>set | Open Addressing | Uses "perturbation" probing scheme (modified linear probing) |
| C# | Dictionary<T,K><br>HashSet<T> | Open Addressing | Uses linear probing |

# Fonction de hachage

- Une fonction de hachage permet de transformer une clé en une valeur de hachage (un index), donnant ainsi la position d'un élément dans le tableau
- Si la clé n'est pas un entier naturel, il faut trouver un moyen de la considérer comme un entier naturel.
- si la clé est de type Chaine de caractères, on peut ajouter la position dans l'alphabet de chaque lettre pour obtenir un entier naturel. Il suffit ensuite de transformer cet entier en index par différentes façons. Par exemple, en prenant le reste de la division de l'entier par le nombre d'éléments dans le tableau.

# Fonction de hachage

- Fonction de hachage de Daniel J. Bernstein :

```
entier hash(str[]) { // str est une chaine de
caractères
    hash = 5381
    i  ← 0
    tant que(str[i] ≠'\0') {
          // hash=hash*33+str[i]
          hash ← ((hash << 5)+hash)+str[i]
          i ← i + 1
    }
    renvoyer hash
}
```

# Facteur de charge(Load factor)

- Le facteur de compression (load factor) qui est la proportion d'alvéoles utilisées dans une table de hachage est une indication critique de ses performances.

$$\text{Load factor} = \frac{n}{m},$$

- n est le nombre de paires clé–valeur ;
- m est le nombre d'alvéoles(slots).

# Facteur de charge(Load factor)

Un facteur de charge faible (par exemple, $\alpha < 0,5$) signifie plus d'espace, moins de collisions et des opérations plus rapides.

Un facteur de charge élevé (par exemple, $\alpha > 0,8$) signifie moins d'espace, plus de collisions et des opérations plus lentes.

# Facteur de charge(Load factor)

La plupart des implémentations de tables de hachage redimensionnent automatiquement le tableau lorsque le facteur de charge dépasse un certain seuil (par exemple, 0,75). Il s'agit d'une opération coûteuse en O(n), mais elle est effectuée rarement pour maintenir la performance moyenne en O(1).

# Python - dict & set

**.Dictionary (key-value pairs)**

```
student_grades = {"alice": 95, "bob": 87}
student_grades["charlie"] = 92          # Insert
grade = student_grades["alice"]         # Lookup
if "bob" in student_grades:             # Check
        print("Found!")
```

**.Set (unique elements)**

```
unique_names = {"alice", "bob"}
unique_names.add("charlie")             # Insert
if "alice" in unique_names:             # Check
```

- print("Exists!")

# Java - HashMap & HashSet

- **HashMap (key-value pairs)**

```
HashMap<String, Integer> studentGrades = new HashMap<>();
studentGrades.put("alice", 95);         // Insert
studentGrades.put("bob", 87);
int grade = studentGrades.get("alice"); // Lookup
if (studentGrades.containsKey("bob")) { // Check
    System.out.println("Found!");}
```

- **HashSet (unique elements)**

```
HashSet<String> uniqueNames = new HashSet<>();
uniqueNames.add("alice");               // Insert
uniqueNames.add("bob");
if (uniqueNames.contains("alice")) {    // Check
    System.out.println("Exists!");}
```

# Hash Tabale

**Django :request.POST**

```
def view(request):
    username = request.POST["username"]  # Hash table lookup!
    email = request.POST["email"]        # Hash table lookup!
```

**Flask :session**

```
@app.route("/")
def home():
    session["user_id"] = 123  # Hash table storage!
```

**JavaScript : objects!**

```
localStorage.setItem("token", "abc123");  // Hash table
```