UNIVERSITE MOHAMMED V ÉCOLE NATIONALE SUPÉRIEURE
D'INFORMATIQUE ET D'ANALYSE DES SYSTÈMES
ENSIAS - Rabat

Filière Sécurité des Systèmes d'Information

Sécurité des systèmes

Compte rendu TP1 : Test d'intégrité

Réalisé par :
SABIR YASSINE

Année Universitaire 2022 - 2023

1 Intégrité

L'intégrité fait référence à l'exactitude et à la cohérence des données, ainsi qu'à la garantie que les données n'ont pas été modifiées de manière illicite ou altérées. C'est un aspect important de la sécurité des données et de la protection des données, car l'intégrité des données est cruciale pour leur fiabilité et leur utilité.

Il existe plusieurs méthodes et technologies utilisées pour garantir l'intégrité des données, telles que les signatures numériques, les codes de correction d'erreurs et les fonctions de hachage, Ces méthodes peuvent être utilisées pour détecter les modifications apportées aux données et empêcher les modifications non autorisées, maintenant ainsi l'intégrité des données au fil du temps.

Dans la suite on va voir un exemple de test d'intégrité basé sur les fonctions de hachage "SHA-1" et "MD5".

2 Fonctions de hachage

Une fonction de hachage est une fonction mathématique qui prend en entrée une donnée quelconque généralement de très grande taille et renvoie une valeur de taille fixe appelée digest ou empreinte. Les fonctions de hachage sont conçues pour être rapides pour la création de l'empreinte, mais lentes pour la récupération des données d'origine à partir de l'empreinte. Cela les rend utiles pour vérifier l'intégrité des données, car tout changement de la donnée d'origine entraînera un changement de l'empreinte générée.

Il existe plusieurs fonctions de hachage utilisés pour identifié les données, parmi ces fonctions en trouve :

2.1 La fonction de hachage "SHA-1"

SHA-1 (Secure Hash Algorithm 1) est une fonction de hachage qui a été largement utilisée pour garantir l'intégrité des données et des applications. Elle a été publiée en 1995 par l'Agence de sécurité nationale des États-Unis (NSA) et est considérée comme une norme de l'industrie pour les fonctions de hachage. Elle prend en entrée un message de taille quelconque et elle produit un résultat empreinte de 160 bits (20 octets), habituellement représenté par un nombre hexadécimal de 40 caractères.

2.2 La fonction de hachage "MD5"

MD5 (Message-Digest Algorithm 5) est une fonction de hachage conçue par Ronald Rivest en 1991 pour améliorer les performances et la sécurité de la fonction de hachage antérieure "MD4", elle prend en entrée un message de n'importe quelle taille et génère une empreinte

de 128 bits. Cette fonction de hachage est connue par sa rapidité d'exécution, ce qui la rend populaire dans le cas des applications qui nécessitent de grandes quantités de traitement de données.

3 Test d'intégrité

Cette partie est réserver au développement d'une interface graphique qui permet de tester l'intégrité d'un message. L'interface offre aux utilisateurs la possibilité d'insérer deux messages, il s'agit du message originale et un message de vérification utilisé pour vérifier l'intégrité du premier message.

3.1 Développement de l'interface graphique

Pour ce qui concerne le développement de l'interface graphique, nous avons utilisé la bibliothèque **java.swing** de **Java**, notamment les objets **JFrame** pour crée la fenêtre qui va contenir l'interface et les objets qu'on va ajouter dans la suite, **JButton** pour créer les différents boutons de l'interface, **JPanel** pour diviser l'interface en des petites interface chacun et conçue d'afficher un contenue spécifique (par exemple JPanel qui contient le message et JPanel qui contient le message à vérifier), **JLabel** permet d'attribuer un nom aux différents composantes de l'interface, **JTextField** permet d'offrir un conteneur qui va contenir le message texte de l'utilisateur, **JFileChooser** qui permet au utilisateur la possibilité de choisir un fichier et puis et objet va contenir le chemin du fichier sélectionné et **JComboBox** qui permet de générer un dropmenu qui va contenir les deux fonctions de hachage pour que l'utilisateur puisse choisir une parmi les deux.

3.2 Fonctions de hachage

Pour ce qui concerne les fonctions de hachage que nous avons utilisé, nous avons utilisé les deux fonctions "SHA-1" et "MD5", pour l'implementation des deux fonctions dans Java, nous avons utiliser la bibliothèque **java.security** notamment l'objet **MessageDigest** qui permet de générer une instance du fonction de hachage souhaité, cette instance a une méthode appelé "digest" qui permet de génère l'empreint sous forme d'un tableau d'octets d'un tableau d'octets, pour cela nous devons transformer le message texte en un tableau d'octets, ce qui justifie l'utilisation du méthode "getBytes" qui effectue la tâche souhaité. Puisque nous avons l'habitude que l'empreinte d'un message est présenter en hexadécimale, nous avons utilisé la fonction "toHexString" qui donne la représentation hexadécimale d'un octet donné en entrée.

```
public static String SHA1(String text) throws Exception {
      MessageDigest digest = MessageDigest.getInstance("SHA-1");
      byte[] hash = digest.digest(text.getBytes("UTF-8"));
      StringBuilder hexString = new StringBuilder();

      for (int i = 0; i < hash.length; i++) {
          String hex = Integer.toHexString(0×ff & hash[i]);
          if (hex.length() == 1) hexString.append('0');
          hexString.append(hex);
      }

      return hexString.toString();
  }
```

FIGURE 1 – Fonction qui calcul l'empreinte d'un texte en utilisant la fonction de hachage "SHA-1"

. La fonction qui calcule l'empreinte d'un texte en utilisant la fonction de hachage "MD5" a la même syntaxe comme celle qui calcule l'empreint avec "SHA-1", mais la différence entre les deux c'est que dans la méthode "getInstance" de l'objet **MessageDigest** au lieu d'écrire "SHA-1" d'écrire "MD5" comme indiqué dans la figure ci-dessous.

```
public static String MD5(String text) throws Exception {
      MessageDigest digest = MessageDigest.getInstance("MD5");
      byte[] hash = digest.digest(text.getBytes("UTF-8"));
      StringBuilder hexString = new StringBuilder();

      for (int i = 0; i < hash.length; i++) {
          String hex = Integer.toHexString(0×ff & hash[i]);
          if (hex.length() == 1) hexString.append('0');
          hexString.append(hex);
      }

      return hexString.toString();
  }
```

FIGURE 2 – Fonction qui calcul l'empreinte d'un texte en utilisant la fonction de hachage "MD5"

.

Pour ce qui concerne le calcule de l'empreinte d'un fichier, en fait la fonction qui effectue cette tâche utilise la même méthodologie que celle qui calcule l'empreint d'un texte, la seule différence que la fonction prend en entrée le chemin du fichier, en fait en utilise la bibliothèque **java.io** notamment l'objet **FileInputStream**, qui va stocker le contenue du fichier, puis on parcours cette objet octet par octet et on ajout ce dernier a un objet de type **MessageDigest**, et le reste est similaire au fonction qui calcule l'empreinte d'un texte.
Voici la fonction qui calcule l'empreinte d'un fichier en utilisant la fonction de hachage "SHA-1".

```
public static String FileSHA1(String fileName) {
      try {
          MessageDigest digest = MessageDigest.getInstance("SHA-1");
          FileInputStream inputStream = new FileInputStream(fileName);
          byte[] bytes = new byte[1024];
          int bytesRead;

          while ((bytesRead = inputStream.read(bytes)) ≠ -1) {
              digest.update(bytes, 0, bytesRead);
          }

          byte[] hash = digest.digest();
          StringBuilder hexString = new StringBuilder();

          for (byte aHash : hash) {
              String hex = Integer.toHexString(0×ff & aHash);
              if (hex.length() == 1) {
                  hexString.append('0');
              }
              hexString.append(hex);
          }

          inputStream.close();
          return hexString.toString();
      } catch (Exception e) {
        return "Erreur de calcul de Hash";
      }
}
```

FIGURE 3 – Fonction qui calcul l'empreinte d'un Fichier en utilisant la fonction de hachage "SHA-1"

. Notons qu'un message indiquant qu'un erreur est rencontré lors du calcul de l'empreint, en général cette erreur on trouve si nous voulons calculer l'empreinte d'un fichier sans le sélectionner comme indiquer dans la figure suivante.

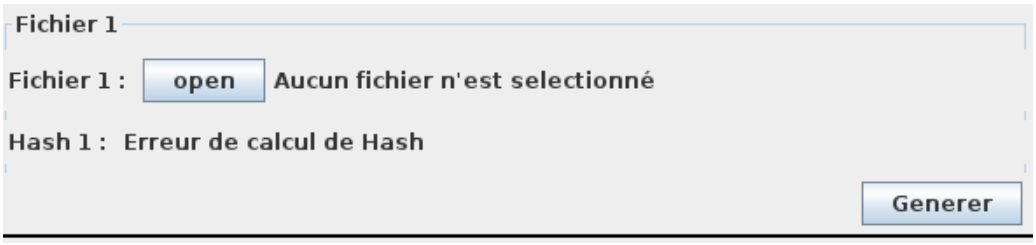

FIGURE 4 – Erreur de calcul de l'empreinte

. Pour ce qui concerne la fonction basée sur le fonction "MD5", il suffit de faire les mêmes modifications du code concernant la fonction qui calcule l'empreinte d'un texte en utilisant la fonction "MD5".

3.3 Gestion des cliques

3.3.1 Boutons "Copier"

Ce bouton permet de copier le contenue de message originale dans le message à vérifié, il y en a deux type de boutons, celle qui permet de copier le message texte, et celui du message qui est sous forme de fichier.

Bouton qui copie le contenue de message texte : une fois en clique sur ce bouton le contenue du message texte originale "Msg1" est dupliqué est stocker dans le message texte de validation "Msg2", comme montré dans la figure suivante .

```
Copier1.addActionListener(new ActionListener(){
  public void actionPerformed(ActionEvent evt) {
    Msg2.setText(Msg1.getText());
  }
});
```

FIGURE 5 – Bouton qui copie le contenue de message texte

Bouton qui copie un message sous forme de fichier : une fois en clique sur ce bouton le chemin du fichier originale "File1Path" est dupliqué est stocker dans le chemin du fichier de validation "File2Path", comme montré dans la figure suivante

```
Copier2.addActionListener(new ActionListener(){
  public void actionPerformed(ActionEvent evt) {
    File2Path.setText(File1Path.getText());
  }
});
```

FIGURE 6 – Bouton qui copie un message sous forme de fichier

.

3.4 Dropmenu des fonctions de hachage

Cette élément à l'utilisateur de choisir une fonction de hachage parmi les deux fonctions "SHA-1" et "MD5". Voici le code qui génère ce dropmenu

```
String[] fonctions = {"SHA-1","MD5"};
JComboBox<String> fonction = new JComboBox<>(fonctions);
```

FIGURE 7 – Dropmenu des fonctions de hachage

.

3.5 Bouton qui permet de choisir un fichier

Une fois on clique sur ce bouton un objet **JFileChooser** est initier, cette objet permet d'ouvrir une fenêtre de chois de fichier comme montré dans le figure suivante.

.

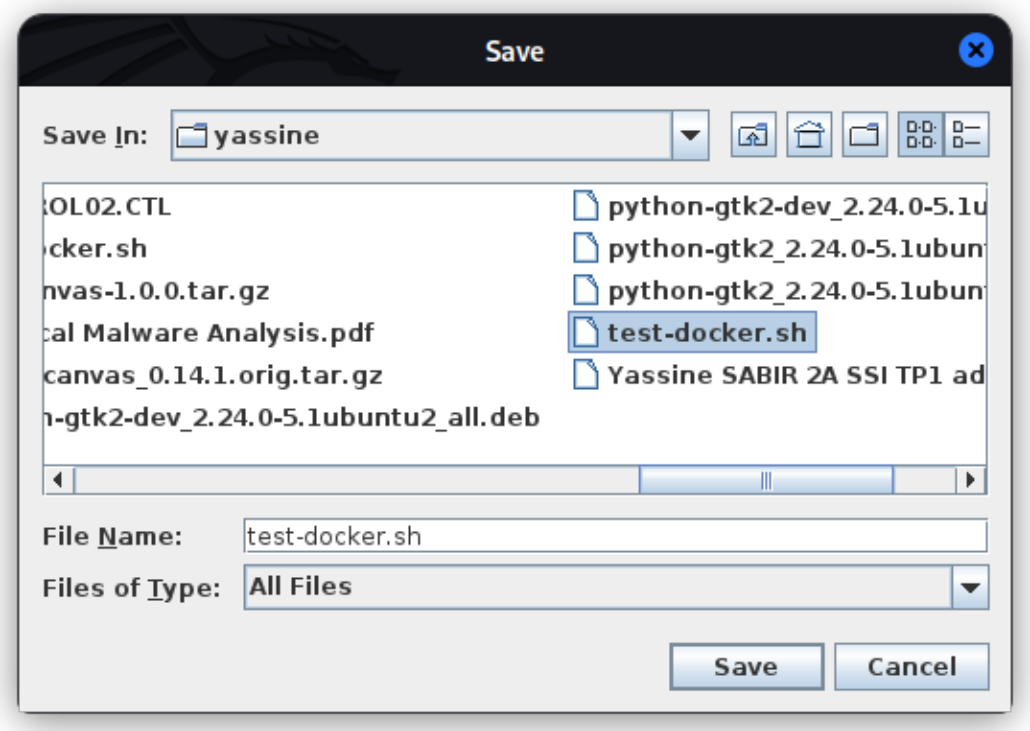

FIGURE 8 – Fenêtre de choix d'un fichier

En plus de ça il permet de stocker le chemin du fichier choisie ce qui va nous aider dans la suite à calculer l'empreinte d'un fichier spécifique.

```
Open1.addActionListener(new ActionListener(){
  public void actionPerformed(ActionEvent evt) {
    JFileChooser J = new JFileChooser(FileSystemView.getFileSystemView().getHomeDirectory());
    int r = J.showSaveDialog(null);
          if (r == JFileChooser.APPROVE_OPTION){
              File1Path.setText(J.getSelectedFile().getAbsolutePath());
          }
          else
              File1Path.setText("Aucun fichier n'est selectionné");
  }
});
```

FIGURE 9 – Bouton qui permet de choisir un fichier

.

3.6 Bouton de Génération de l'empreinte de message

Ce bouton permet de générer l'empreinte du message saisie par l'utilisateur, i y en a deux type de bouton celle qui génère l'empreinte du message texte et celui qui génère l'empreinte du fichier.

Bouton qui génère l'empreinte d'un message texte : une fois on clique sur ce bouton, on prend le message texte stocké dans "Msg1" et calcule son empreinte en prenant en considération

la fonction de hachage choisie par l'utilisateur, il s'agit de nom de fonction stocké dans le dropmenu que nous avons nommé "fonction" et en utilisant les fonctions que nous avons crée au début.

```
Generer1_1.addActionListener(new ActionListener(){
  public void actionPerformed(ActionEvent evt) {
    String message = Msg1.getText();
    if (fonction.getSelectedItem().toString().equalsIgnoreCase("SHA-1")) {
      try {
        Hash1_1.setText(SHA1(message));
      } catch (Exception e) {
      }
    }
    else {
      try {
        Hash1_1.setText(MD5(message));
      } catch (Exception e) {
      }
    }
  }
});
```

FIGURE 10 – Bouton qui génère l'empreinte d'un message texte

. **Bouton qui génère l'empreinte d'un fichier :** une fois on clique sur ce bouton, on prend le chemin du fichier stocker dans la variable "File1Path" et calcule l'empreinte du fichier en prenant en considération la fonction de hachage choisie par l'utilisateur, il s'agit de nom de fonction stocké dans le dropmenu que nous avons nommé "fonction" et en utilisant les fonctions que nous avons crée au début.

```
Generer1_2.addActionListener(new ActionListener(){
  public void actionPerformed(ActionEvent evt) {
    String path = File1Path.getText();
    if (fonction.getSelectedItem().toString().equalsIgnoreCase("SHA-1")) {
      try {
        Hash1_2.setText(FileSHA1(path));
      } catch (Exception e) {
      }
    }
    else {
      try {
        Hash1_2.setText(FileMD5(path));
      } catch (Exception e) {
      }
    }
  }
});
```

FIGURE 11 – Bouton qui génère l'empreinte d'un fichier

.

3.7 Bouton qui vérifie l'intégrité

Il s'agit d'un bouton qui permet de tester l'intégrité du message et renvoyer un message qui indique si l'intégrité est vérifié ou pas. Il y en a deux types de boutons, un qui teste l'intégrité du message texte et l'autre teste l'intégrité du fichier sélectionné.

Bouton qui teste l'intégrité d'un message texte : une fois on clique sur ce bouton, le programme récupère le contenue des deux message, "Msg1" qui est le message originale et "Msg2" qui est le message à vérifié, ensuite en se basant sur la fonction de hachage choisit par l'utilisateur (stocker dans la variable "fonction"), le programme calcul l'empreinte des deux messages, puis il les comparent, si il trouve qu'elles sont identiques, il retourne un message qui indique que l'intégrité est vérifié, dans l'autre cas un message qui indique qu'on a pas d'intégrité est retourné. On remarque que le résultat (message retourné par le programme) est stocké dans un variable nommé "Result", ce variable est celui qui manipule le message a affiché dans la partie résultat qui situe au dessus de l'interface. dans le cas d'exception un message indiquant qu'on doit saisie les deux messages est affiché.

```
Verifie1.addActionListener(new ActionListener(){
  public void actionPerformed(ActionEvent evt) {
    String message1 = Msg1.getText(), message2 = Msg2.getText();
    if (fonction.getSelectedItem().toString().equalsIgnoreCase("SHA-1")) {
      try {
        String hash1 = SHA1(message1);
        String hash2 = SHA1(message2);

        if (hash1.equalsIgnoreCase(hash2)) {
          Result.setText("L'intégrité est verifié");
        }
        else {
          Result.setText("l'intégrité n'est pas verifié");
        }

      } catch (Exception e) {
        Result.setText("Entrer les deux messages");
      }
    }
    else {
      try {
        String hash1 = MD5(message1);
        String hash2 = MD5(message2);

        if (hash1.equalsIgnoreCase(hash2)) {
          Result.setText("L'intégrité est verifié");
        }
        else {
          Result.setText("l'intégrité n'est pas verifié");
        }

      } catch (Exception e) {
        Result.setText("Entrer les deux messages");
      }
    }
  }
});
```

FIGURE 12 – Bouton qui teste l'intégrité d'un message texte

. **Bouton qui teste l'intégrité d'un fichier :** ce bouton joue le même rôle du bouton précédent mais la seule différence entre les deux c'est que celui-ci utilise les chemins des deux fichiers et il utilise les fonctions de calcule de l'empreinte des fichiers, il s'agit des fonction que nous avons développé précédemment. Voici le code des boutons de ce type

```
Verifie2.addActionListener(new ActionListener(){
  public void actionPerformed(ActionEvent evt) {
    String path1 = File1Path.getText(), path2 = File2Path.getText();
    if (fonction.getSelectedItem().toString().equalsIgnoreCase("SHA-1")) {
      try {
        String hash1 = FileSHA1(path1);
        String hash2 = FileSHA1(path2);

        if (hash1.equalsIgnoreCase(hash2)) {
          Result.setText("L'intégrité est verifié");
        }
        else {
          Result.setText("l'intégrité n'est pas verifié");
        }

      } catch (Exception e) {
        Result.setText("Entrer les deux messages");
      }
    }
    else {
      try {
        String hash1 = FileMD5(path1);
        String hash2 = FileMD5(path2);

        if (hash1.equalsIgnoreCase(hash2)) {
          Result.setText("L'intégrité est verifié");
        }
        else {
          Result.setText("l'intégrité n'est pas verifié");
        }

      } catch (Exception e) {
        Result.setText("Entrer les deux messages");
      }
    }
  }
});
```

FIGURE 13 – Bouton qui teste l'intégrité d'un fichier

.

3.8 Vision globale sur l'interface

Le programme prend en entrer deux messages, textes ou bien fichiers, ensuite l'utilisateur à le droit de choisir une fonction de hachage parmi les deux fonctions "SHA-1" et "MD5", à ce moment là l'utilisateur peut générer l'empreint des deux messages, l'empreinte de chaque

message est écrit au dessous de ce dernier.

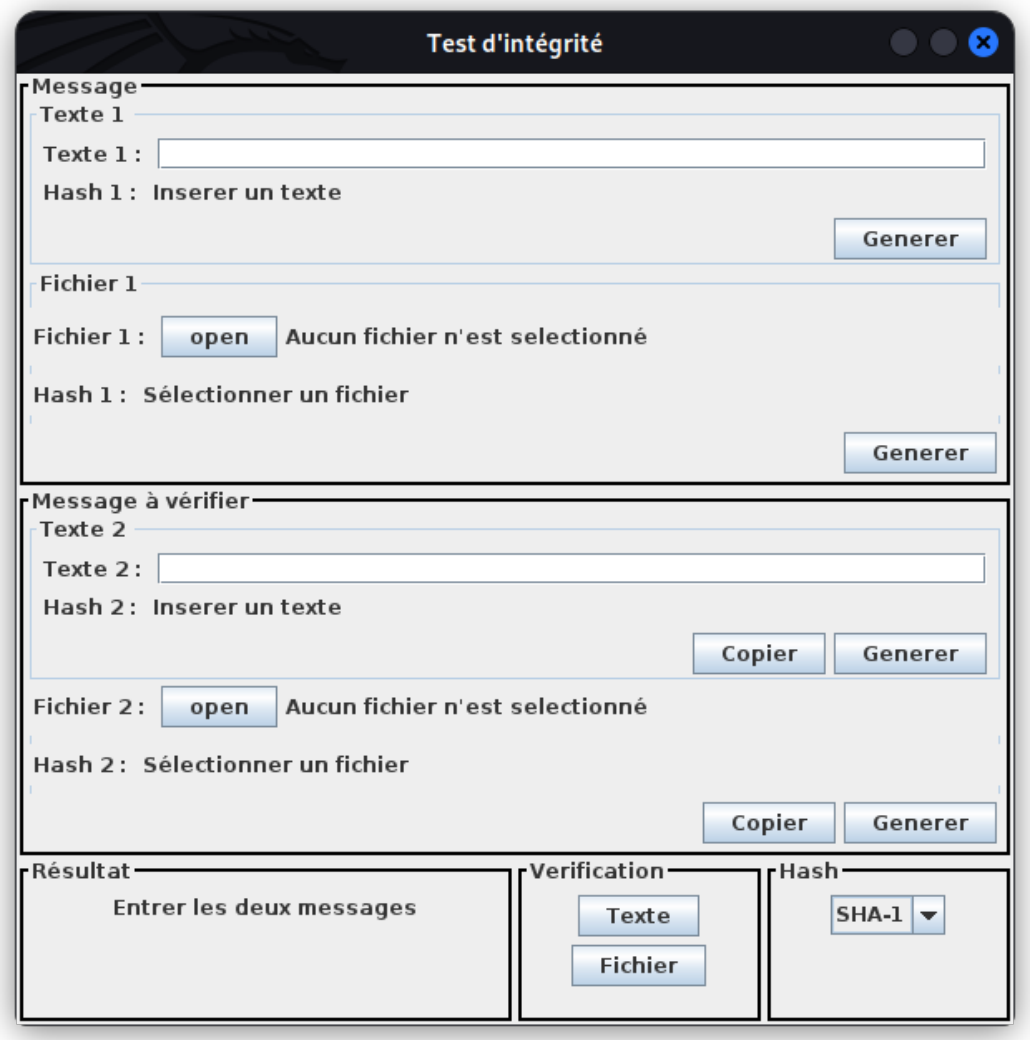

FIGURE 14 – Test d'intégrité

. Voici l'empreinte du mot "Bonsoir" et d'un fichier "test-docker.sh", générés par la fonction "SHA-1"

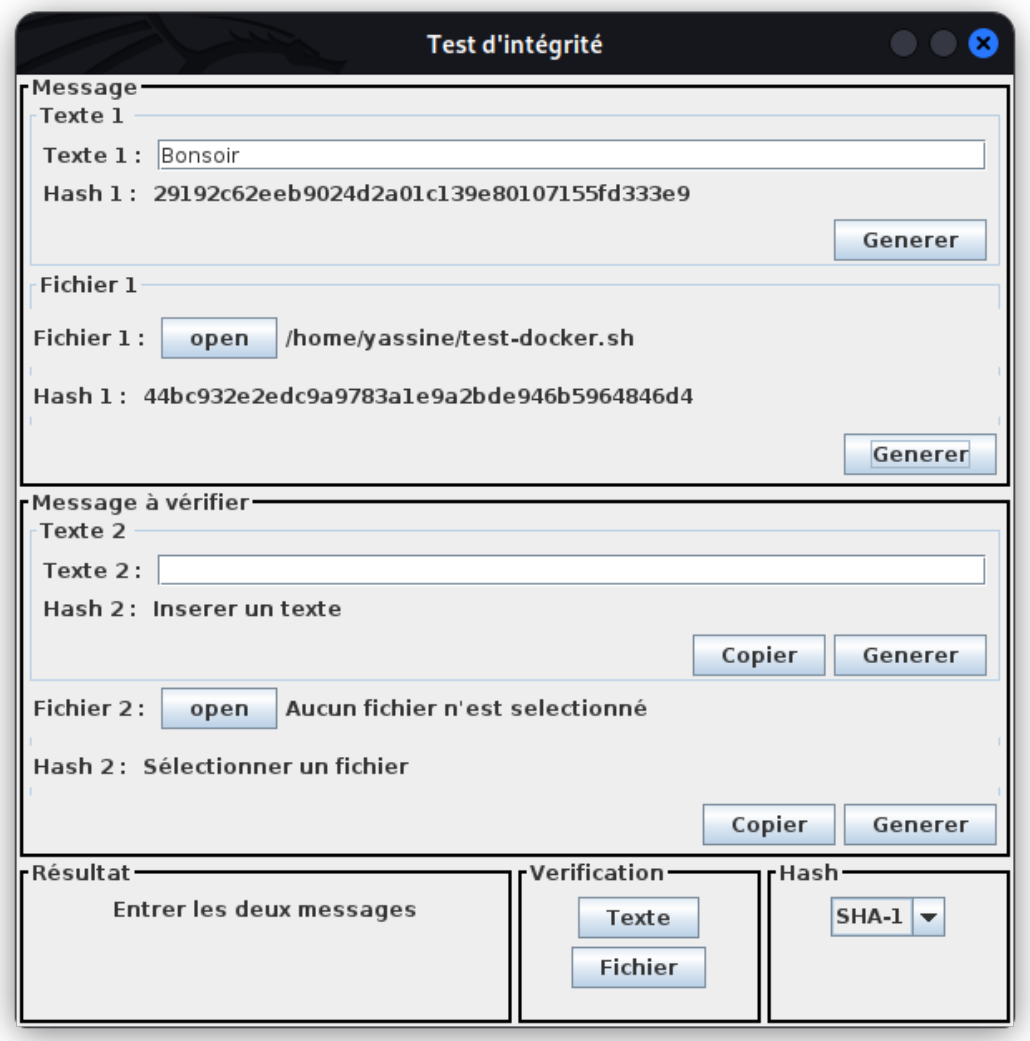

FIGURE 15 – Empreint généré par la fonction "SHA-1"

.

Voici les empreintes des deux messages, mais cette fois elles sont générés par la fonction "MD5"

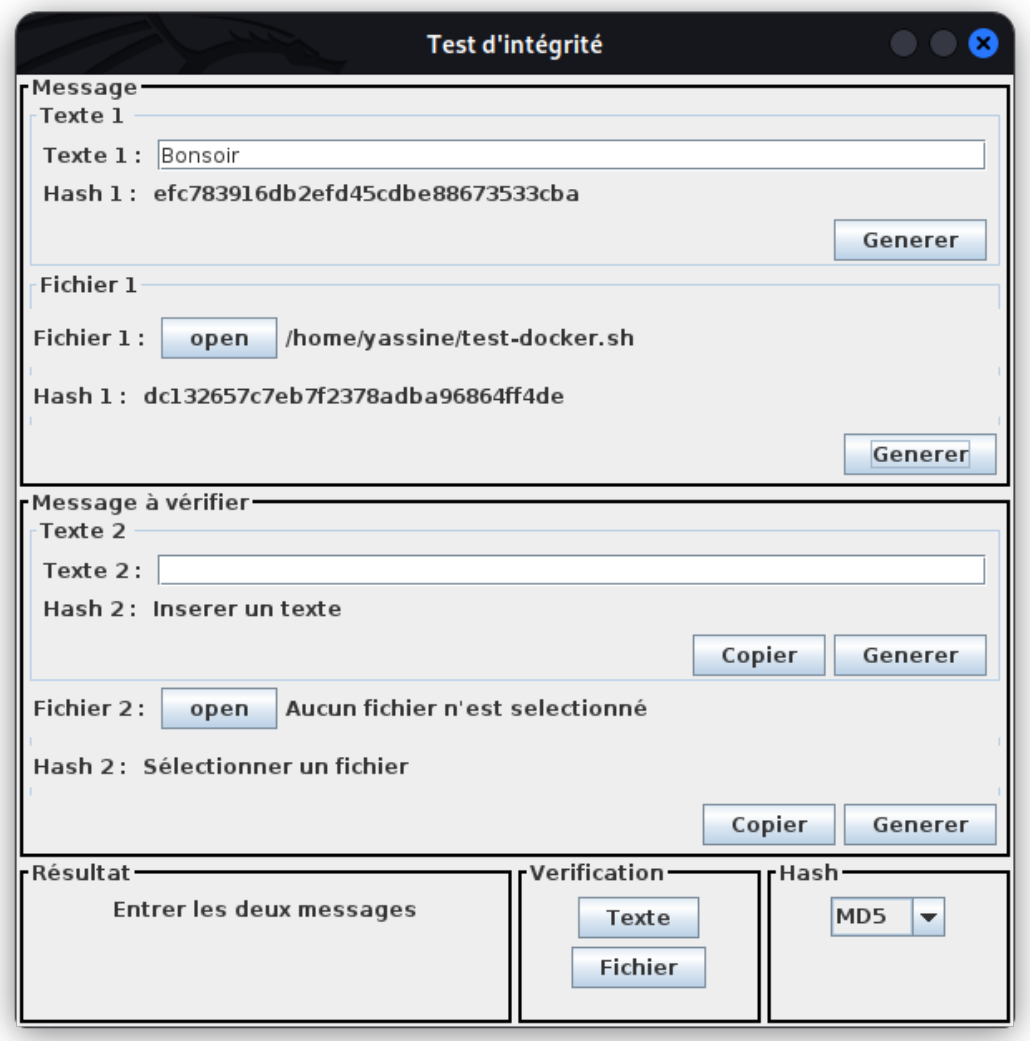

FIGURE 16 – Empreint généré par la fonction "MD5"

.

Si maintenant l'utilisateur veut vérifier l'intégrité du message il choisit la nature du message dans le menu de vérification (texte ou fichier), ensuite le programme calcule l'empreinte des deux machines et les comparent puis il affiche le résultat de l'intégrité dans le champ résultat.

Voici un exemple de message qui ne vérifie pas l'intégrité

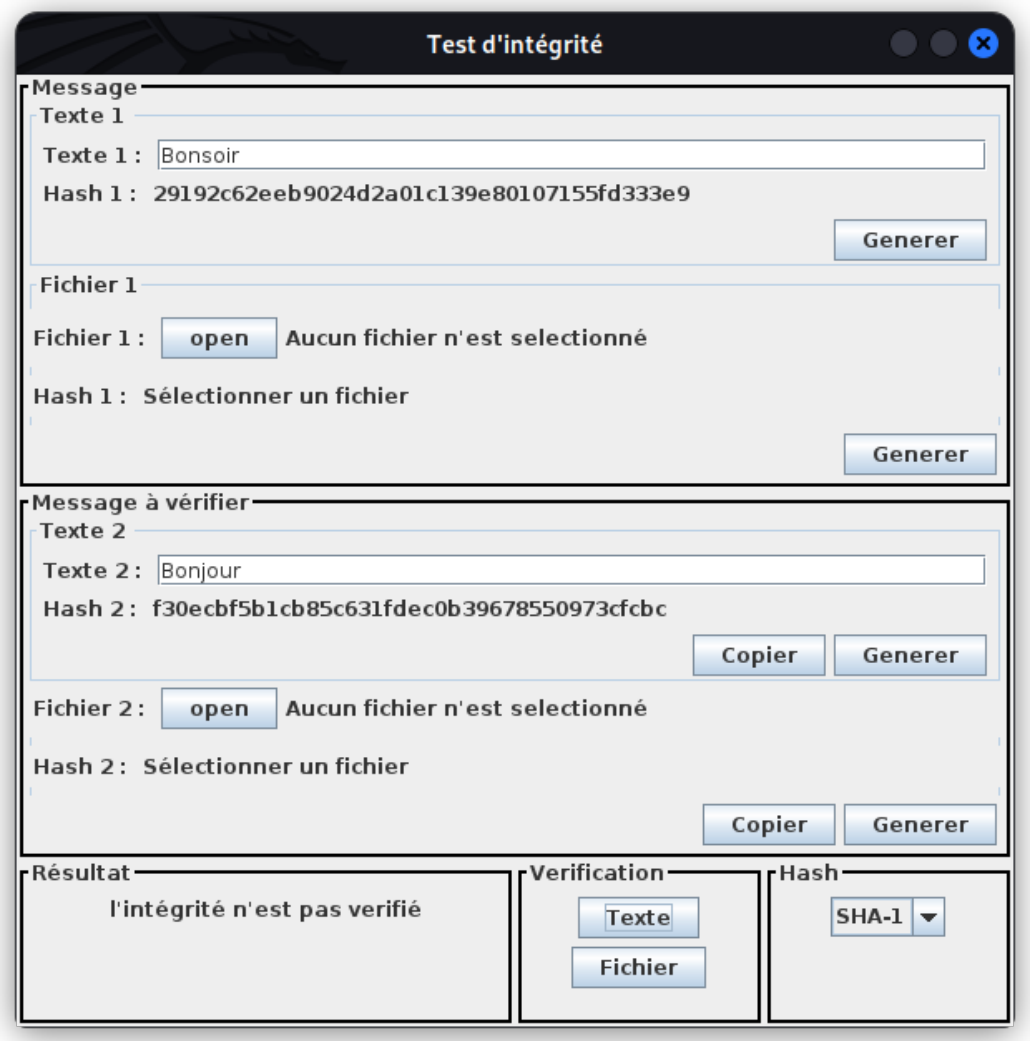

FIGURE 17 – Intégrité n'est pas vérifié

.

Et voila un autre message qui vérifie l'intégrité

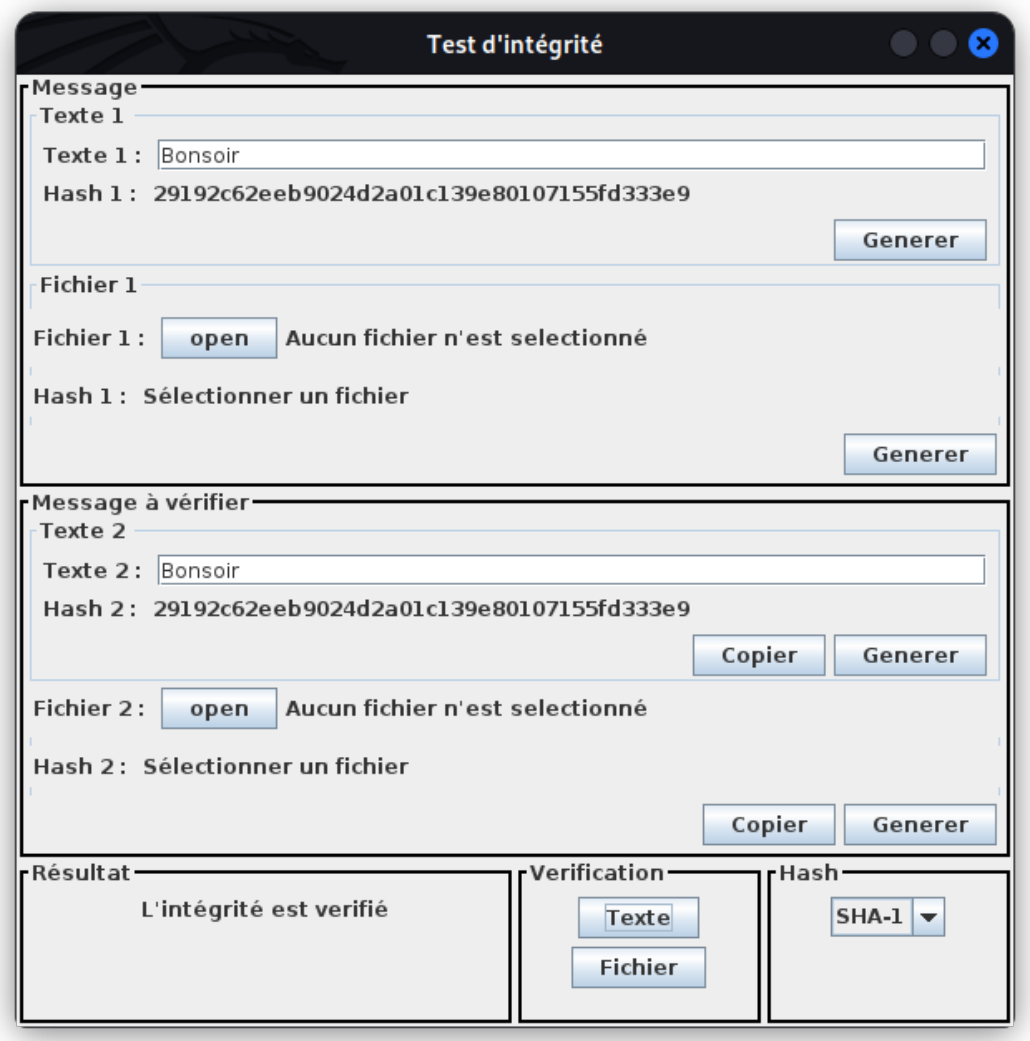

FIGURE 18 – Intégrité vérifié

.