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 TP2 : Test d'authentification et d'intégrité

Réalisé par :
SABIR YASSINE

Année Universitaire 2022 - 2023

1 L'authentification et l'intégrité dans les systèmes d'information

L'authentification est le processus permettant de vérifier l'identité d'un utilisateur ou d'un système informatique. Dans le contexte de la sécurité informatique, l'authentification est utilisée pour vérifier que l'entité qui accède à un système est bien celle qu'elle prétend être. L'authentification peut être réalisée à l'aide de différents mécanismes, tels que les mots de passe, les certificats numériques, les clés d'authentification, etc.

L'intégrité est l'un des principaux concepts de la sécurité informatique, elle désigne la capacité à garantir que les données n'ont pas été altérées ou modifiées de manière non autorisée pendant leur stockage ou leur transmission. Dans le contexte de la sécurité informatique, l'intégrité des données est essentielle pour s'assurer que les données échangées sont fiables et n'ont pas été altérées par une personne malveillante.

Dans le cadre de la vérification de l'authenticité et l'intégrité d'un message, l'authentification est utilisée pour vérifier que le message a bien été envoyé par l'émetteur prétendu, et l'intégrité permet d'assurer que le message n'a pas été altéré pendant sa transmission. Cela peut être réalisé en utilisant des mécanismes de signature numérique tels que "DSA" ou "RSA", qui permettent de garantir que le message n'a pas été altéré et qu'il a bien été émis par l'émetteur prétendu, c'est ce qu'on va voir dans la suite.

2 Signature

La signature numérique est un mécanisme de sécurité informatique qui permet de garantir l'authenticité, l'intégrité et la non-répudiation d'un document ou d'un message électronique. Elle est utilisée pour garantir que le document ou le message n'a pas été altéré depuis sa création et pour identifier l'auteur de la signature.

La signature numérique utilise une clé privée pour générer la signature, et une clé publique pour la vérifier. Les algorithmes de signature numérique tels que DSA (Digital Signature Algorithm) et RSA (Rivest-Shamir-Adleman) sont couramment utilisés pour générer des signatures numériques.

Pour créer une signature numérique, l'auteur de la signature génère un hachage (une empreinte numérique) du document ou du message à signer, puis utilise sa clé privée pour chiffrer le hachage. La signature résultante est envoyée avec le document ou le message.

Pour vérifier la signature, le destinataire utilise la clé publique de l'auteur de la signature pour déchiffrer la signature et obtenir le hachage original. Il calcule ensuite le hachage du document ou du message reçu et compare les deux hachages. Si les hachages correspondent, la signature est considérée comme valide et le document ou le message est considéré comme authentique.

3 Digital Signature Algorithm - DSA

DSA (Digital Signature Algorithm) est un algorithme de signature numérique utilisé pour garantir l'authenticité et l'intégrité des données échangées en ligne. Il a été développé par le National Institute of Standards and Technology (NIST) et publié en 1991.

DSA utilise une combinaison de techniques mathématiques, notamment la théorie des nombres et la cryptographie à clé publique, pour générer des signatures numériques. Il est principalement utilisé pour signer des messages et des documents électroniques, ainsi que pour sécuriser les transactions en ligne.

L'algorithme DSA est basé sur l'utilisation de nombres premiers et de fonctions de hachage cryptographiques. Il utilise une paire de clés : une clé privée qui est gardée secrète par l'utilisateur, et une clé publique qui est largement diffusée et utilisée pour vérifier les signatures générées à l'aide de la clé privée.

Pour créer une signature numérique avec DSA, l'auteur de la signature utilise sa clé privée pour générer une signature à partir d'un hachage cryptographique du message à signer. La signature est ensuite envoyée avec le message signé. Le destinataire utilise la clé publique de l'auteur de la signature pour vérifier l'authenticité de la signature.

4 Rivest Shamir Adleman - RSA

RSA (Rivest-Shamir-Adleman) est un algorithme de cryptographie à clé publique utilisé pour sécuriser les communications en ligne. Il a été inventé en 1977 par Ron Rivest, Adi Shamir et Leonard Adleman au MIT.

RSA utilise une paire de clés : une clé publique qui peut être distribuée largement et utilisée pour chiffrer les données, et une clé privée qui doit être gardée secrète et utilisée pour déchiffrer les données chiffrées.

L'algorithme RSA est basé sur la théorie des nombres et l'arithmétique modulaire. Il utilise une fonction de chiffrement qui dépend d'une paire de nombres premiers distincts et d'un entier appelé exposant de chiffrement. La sécurité de RSA repose sur le fait qu'il est difficile de factoriser un grand nombre composé en ses facteurs premiers.

Pour chiffrer un message avec RSA, l'expéditeur utilise la clé publique du destinataire pour chiffrer le message. Le destinataire utilise sa clé privée pour déchiffrer le message chiffré.

5 Test d'authentification et d'intégrité en utilisant la signature

Cette partie est réserver au développement d'une interface graphique qui permet de vérifier l'authentification et l'intégrité d'un texte. L'interface offre aux utilisateurs la possibilité d'insérer

un texte, choisir l'algorithme de génération de signature, il s'agit des deux algorithmes "DSA" et "RSA", en plus de ça la signature générer par l'algorithme choisie est affiché dans la place réservé à cette fonctionnalité, l'utilisateur peut ensuite modifier insérer une autre signature et de tester si elle correspond au texte inséré.
Voici une image qui montre les fonctionnalité de l'interface

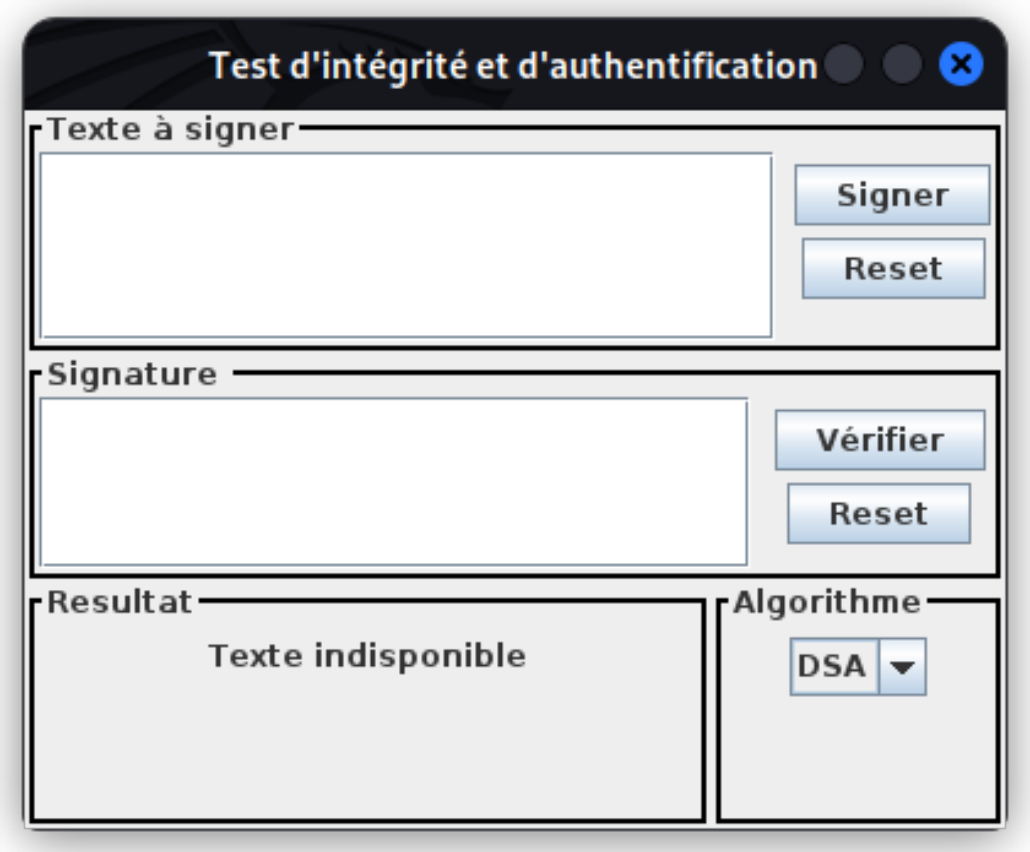

FIGURE 1 – Teste d'authentification et d'intégrité

On peut diviser les fonctionnalités que l'interface offre :

5.1 Générer la signature d'un texte

Dans cette partie nous supposons que l'utilisateur a choisie un algorithme de génération de signature parmi les deux algorithmes proposés par l'interface, l'utilisateur peut insérer du texte dans la partie réservé à cela, et si il clique sur le bouton signer la signature de ce texte en utilisons l'algorithme choisie par l'utilisateur est générer et affiché dans la partie "Signature" de l'interface.
Pour ce qui concerne le code java qui génère une signature DSA par exemple d'un texte donnée, nos avons utilisé la fonction suivante :

```java
public static String Signature_DSA(String Texte, KeyPair keyPair) throws Exception {

  // Créer une signature DSA
  Signature dsa = Signature.getInstance("SHA256withDSA");
  dsa.initSign(keyPair.getPrivate());
  dsa.update(Texte.getBytes("UTF-8"));
  byte[] signature = dsa.sign();


  // Convertir la signature en base64 pour l'affichage
  String encodedSignature = Base64.getEncoder().encodeToString(signature);
  return encodedSignature;
}
```

FIGURE 2 – Fonction qui génère signature DSA d'un texte

cette fonction prend en entrer un texte et un objet de type **KeyPair**, ensuite il prend le texte que nous avons donnée et il le stock dans un objet **dsa** pour ensuite générer la signature de ce texte en utilisant la clé privé inclut dans l'objet KeyPair que nous avons donnée comme entrée de cette fonction, nous avons choisit pour ce qui concerne la fonction de hachage utilisé dans la génération du signature nous avons choisit la fonction **SHA256**, on remarque que la signature généré est sous forme de bytes donc nous devons la convertir pour qu'elle sera lisible Pour ce qui concerne l'objet **KeyPair**, il permet de stocker la clé publique et la clé privé que nous devons utiliser dans la génération et la vérification de la signature généré par l'algorithme "DSA", voici le code utilisé pour générer cet objet

```java
//DSA
KeyPairGenerator keyGen_DSA = KeyPairGenerator.getInstance("DSA");
SecureRandom random1 = SecureRandom.getInstanceStrong();
keyGen_DSA.initialize(1024, random1);
KeyPair keyPair_DSA = keyGen_DSA.generateKeyPair();
```

FIGURE 3 – Génération du clé privé et clé publique "DSA"

Pour ce qui concerne la fonction qui calcule la signature d'un texte en utilisant l'algorithme RSA, cette fonction à la même structure que celle de DSA, les deux figures suivantes montrent cette fonction ainsi que le programme qui génere la clé privé et la clé public utilisés dans cette fonction.

```java
public static String Signature_RSA(String Texte, KeyPair keyPair) throws Exception {

  // Créer une signature DSA
  Signature rsa = Signature.getInstance("SHA256withRSA");
  rsa.initSign(keyPair.getPrivate());
  rsa.update(Texte.getBytes("UTF-8"));
  byte[] signature = rsa.sign();

  // Convertir la signature en base64 pour l'affichage
  String encodedSignature = Base64.getEncoder().encodeToString(signature);
  return encodedSignature;
}
```

FIGURE 4 – Fonction qui génère signature RSA d'un texte

```java
//RSA
KeyPairGenerator keyGen_RSA = KeyPairGenerator.getInstance("RSA");
SecureRandom random2 = SecureRandom.getInstanceStrong();
keyGen_RSA.initialize(1024, random2);
KeyPair keyPair_RSA = keyGen_RSA.generateKeyPair();
```

FIGURE 5 – Génération du clé privé et clé publique "RSA"

Pour générer la signature du texte en se basant sur l'algorithme choisit par l'utilisateur, nous avons utilisé le code décrit dans la figure suivante qui gère les cliques de ce bouton.
En effet une fois ce bouton est cliqué, le programme vérifie qu'elle algorithme de génération de signature l'utilisateur a choisis, ce choix est stocké dans l'objet "signature", ensuite le programme calcule la signature de ce texte est il l'affiche dans l'objet "SignatureTexte".

```
signer.addActionListener(new ActionListener(){
  public void actionPerformed(ActionEvent evt) {
    String texte = Texte.getText();
    if (signature.getSelectedItem().toString().equalsIgnoreCase("DSA")) {
      try {
        String Signature_texte = Signature_DSA(texte, keyPair_DSA);
        SignatureTexte.setText(Signature_texte);
      }
      catch(Exception e){

      }
    }
    else {
      try {
        String Signature_texte = Signature_RSA(texte, keyPair_RSA);
        SignatureTexte.setText(Signature_texte);
      }
      catch(Exception e){

      }
    }
  }
});
```

FIGURE 6 – Bouton qui génère la signature

Voici un exemple de génération de signature en utilisant l'algorithme DSA

FIGURE 7 – Exemple signature DSA

Voici le résultat du même exemple mais en utilisant RSA

FIGURE 8 – Exemple signature RSA

5.2 Validation

Dans la partie validation nous devons utilisé la clé publique utilisé dans la génération de la signature, ensuite de déchiffrer la signature trouver, ensuite nous gênèrent l'empreinte du texte que nous avons on utilisons la même fonction de hachage qui est **SHA256**, et comparer à la fin l'empreinte du texte que nous avons généré avec le résultat du déchiffrement de la signature.
Pour ce qui concerne la vérification de signature généré par l'algorithme DSA, nous avons utilisé la fonction suivante

```
public static boolean verifySignature_DSA(String data, byte[] signature, PublicKey publicKey) throws Exception {
      // Créer une signature DSA
      Signature dsa = Signature.getInstance("SHA256withDSA");
      dsa.initVerify(publicKey);
      dsa.update(data.getBytes("UTF-8"));

      // Vérifier la signature
      boolean verified = dsa.verify(signature);

      return verified;
}
```

FIGURE 9 – Vérification de la signature généré par DSA

Cette fonction prend en entrée le texte origine, la signature que nous voulons valider et la clé publique qu'on va utilisé dans le déchiffrement de la signature. La fonction prend le texte à l'entrée puis il le stocke dans un objet dsa, ensuite dans cette objet on déchiffre la signature donnée en entrée avec la clé publique donnée aussi en entrée, à la fin en compare le résultat de cet déchiffrement avec l'empreinte du texte générer par la fonction **SHA256** et en retourne une valeur booléenne.
Pour ce qui concerne la validation des signature généré par RSA nous avons utilisé la fonction suivante qui est similaire que celle de DSA.

```
public static boolean verifySignature_RSA(String data, byte[] signature, PublicKey publicKey) throws Exception {
      // Créer une signature DSA
      Signature dsa = Signature.getInstance("SHA256withRSA");
      dsa.initVerify(publicKey);
      dsa.update(data.getBytes("UTF-8"));

      // Vérifier la signature
      boolean verified = dsa.verify(signature);

      return verified;
}
```

FIGURE 10 – Vérification de la signature généré par DSA

Pour ce qui concerne le bouton de vérification, ce permet de vérifier si la signature qu'on a correspond bien au texte que nous avons saisit, il se base dans son fonctionnement sur les fonction de vérification que nous avons développé et de produire à fin un résultat qui indique si la vérification est réussie ou non. La figure suivante présente le code qui permet de génère des bouton qui font le même fonctionnement.

```
verifier.addActionListener(new ActionListener(){
  public void actionPerformed(ActionEvent evt) {
    String texte = Texte.getText();
    String Sign = SignatureTexte.getText();
    if (signature.getSelectedItem().toString().equalsIgnoreCase("DSA")) {
      try {
        if (verifySignature_DSA(texte, Base64.getDecoder().decode(Sign), keyPair_DSA.getPublic())) {
          resultat.setText("Vérification réussie de la signature");
        }
        else{
          resultat.setText("La signature n'est pas vérifiée");
        }
      }
      catch(Exception e){
        resultat.setText("Texte indisponible");
      }
    }
    else {
      try {
        if (verifySignature_RSA(texte, Base64.getDecoder().decode(Sign), keyPair_RSA.getPublic())) {
          resultat.setText("Vérification réussie de la signature");
        }
        else{
          resultat.setText("La signature n'est pas vérifiée");
        }
      }
      catch(Exception e){
        resultat.setText("Texte indisponible");
      }
    }
  }
});
```

FIGURE 11 – Vérification de la signature généré par DSA

Voici un exemple de validation réussie de la signature

FIGURE 12 – Signature valide

Voici un exemple d'échec de vérification de signature

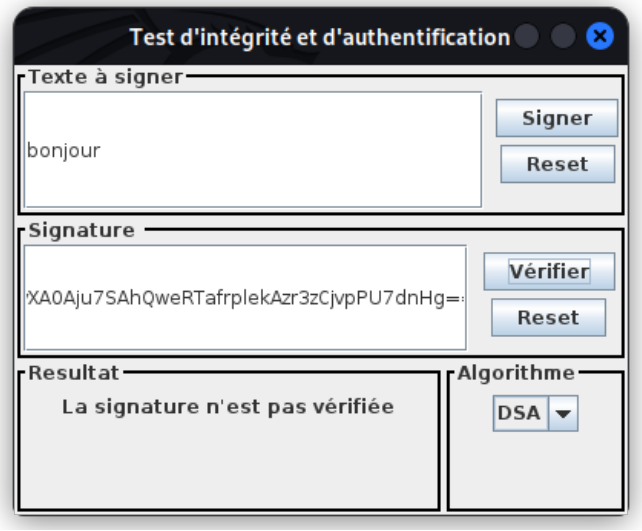

FIGURE 13 – Echec de validation de signature

5.3 Boutons de réinitialisation

Les boutons de réinitialisation ou reset permet de libérer le tampon de texte ou de signature de notre interface.