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 TP3 : Certificat numérique

Réalisé par :
SABIR YASSINE

Année Universitaire 2022 - 2023

1 Certificat numérique

Un certificat numérique, également appelé certificat électronique, est un document électronique utilisé pour identifier de manière sécurisée les personnes, les ordinateurs, les serveurs et les sites web. Il est utilisé pour garantir l'authenticité des informations transmises sur Internet et pour assurer la confidentialité des données échangées.

Un certificat numérique est délivré par une autorité de certification (AC) qui vérifie l'identité de la personne ou de l'entité à qui le certificat est délivré. Lorsqu'un utilisateur accède à un site web sécurisé, le navigateur web vérifie la validité du certificat numérique du site web en contactant l'autorité de certification. Si le certificat est valide, la communication entre l'utilisateur et le site web est sécurisée.

Les certificats numériques sont utilisés dans de nombreuses applications, telles que le commerce électronique, la banque en ligne, la signature électronique, la messagerie sécurisée, l'accès aux réseaux privés virtuels (VPN) et aux services cloud.

2 Les composants du certificat numérique

Un certificat numérique contient plusieurs informations, notamment :

— **Le nom du titulaire du certificat :** Il s'agit du nom de l'entité à qui le certificat est délivré, qu'il s'agisse d'une personne ou d'une organisation.

— **La clé publique du titulaire :** Il s'agit d'une clé de chiffrement qui peut être utilisée par d'autres personnes pour envoyer des messages chiffrés au titulaire du certificat.

— **L'identité de l'autorité de certification (CA) :** C'est l'entité qui a émis le certificat et qui atteste de l'authenticité du titulaire.

— **La période de validité du certificat :** C'est la période pendant laquelle le certificat est considéré comme valide. Une fois cette période expirée, le certificat doit être renouvelé.

— **Le numéro de série du certificat :** C'est un numéro unique qui identifie de manière unique le certificat dans la base de données de l'autorité de certification.

— **La signature numérique de l'autorité de certification :** Cette signature garantit l'authenticité du certificat en attestant que le certificat a été émis par l'autorité de certification et qu'il n'a pas été modifié depuis sa création.

— **Les politiques de sécurité :** Il s'agit de règles et de procédures qui ont été suivies pour garantir la sécurité de l'émission du certificat.

Toutes ces informations sont stockées dans un format standardisé appelé X.509, qui est utilisé pour la gestion des certificats numériques. Les certificats numériques sont largement utilisés pour sécuriser les communications en ligne et les transactions financières, entre autres utilisations.

3 Comment les certificats numériques sont générés ?

Les certificats numériques sont générés par une autorité de certification (CA - Certificate Authority) qui est une entité de confiance chargée de vérifier l'identité d'une personne ou d'une organisation avant de lui délivrer un certificat numérique.

Pour générer un certificat numérique, une autorité de certification doit tout d'abord vérifier l'identité du demandeur. Pour cela, elle peut exiger des preuves d'identité telles qu'une carte d'identité, un passeport, un permis de conduire ou d'autres documents officiels. Une fois l'identité vérifiée, l'autorité de certification crée une paire de clés cryptographiques (une clé publique et une clé privée) pour le demandeur.

Le certificat numérique contient des informations sur le demandeur, telles que son nom, son adresse e-mail et sa clé publique. Le certificat est ensuite signé numériquement par l'autorité de certification, ce qui garantit l'authenticité et l'intégrité du certificat.

Une fois que le certificat est émis, il peut être utilisé pour des opérations telles que la signature électronique, l'authentification de l'identité d'un utilisateur ou d'un serveur, et le chiffrement des communications. Les certificats numériques ont un rôle important dans la sécurité des communications sur Internet, en particulier pour les transactions financières et les communications sensibles.

4 La signature numérique

Nous avons vu dans le travail pratique précèdent comment la signature numérique permet d'assurer l'authentification et l'intégrité dans les systèmes d'information, en fait l'authentification est le processus qui permet de 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.

Nous avons aussi vu c'est quoi la signature numérique et comment elle permet de garantir l'authenticité, l'intégrité et la non-répudiation d'un document ou d'un message électronique, et nos avons présenté deux exemple d'algorithme de génération de signature, il s'agit des deux algorithmes **DSA** et **RSA**. 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.
En fait **DSA (Digital Signature Algorithm)** est un algorithme de signature numérique basé sur la théorie mathématique des nombres premiers et des fonctions de hachage. Il est utilisé pour garantir l'authenticité, l'intégrité et la non-répudiation des données électroniques. L'algorithme DSA génère une paire de clés cryptographiques, publique et privée, qui sont utilisées pour signer et vérifier les données. La clé privée est utilisée pour signer les données, tandis que la clé publique est utilisée pour vérifier la signature. L'algorithme DSA est largement utilisé dans les applications de sécurité telles que les transactions financières, les communications en ligne et les échanges de courrier électronique pour garantir la sécurité et la confidentialité des données. Et **RSA** est un algorithme de cryptographie à clé publique inventé par **Ron Rivest, Adi Shamir et Leonard Adleman** en 1977. Il est largement utilisé pour la sécurisation des communications électroniques, notamment pour le chiffrement et la signature électronique. L'algorithme est basé sur la difficulté mathématique de factoriser de grands nombres entiers en produits de nombres premiers, ce qui rend la clé de chiffrement pratiquement impossible à casser pour un attaquant. RSA utilise une paire de clés, une clé publique pour le chiffrement des données et une clé privée pour le déchiffrement des données.

5 Génération des certificats numériques

Cette partie est réserver au développement d'une interface graphique qui permet de vérifier l'authentification et l'intégrité d'un texte aisi de génerer un certificat numérique. 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

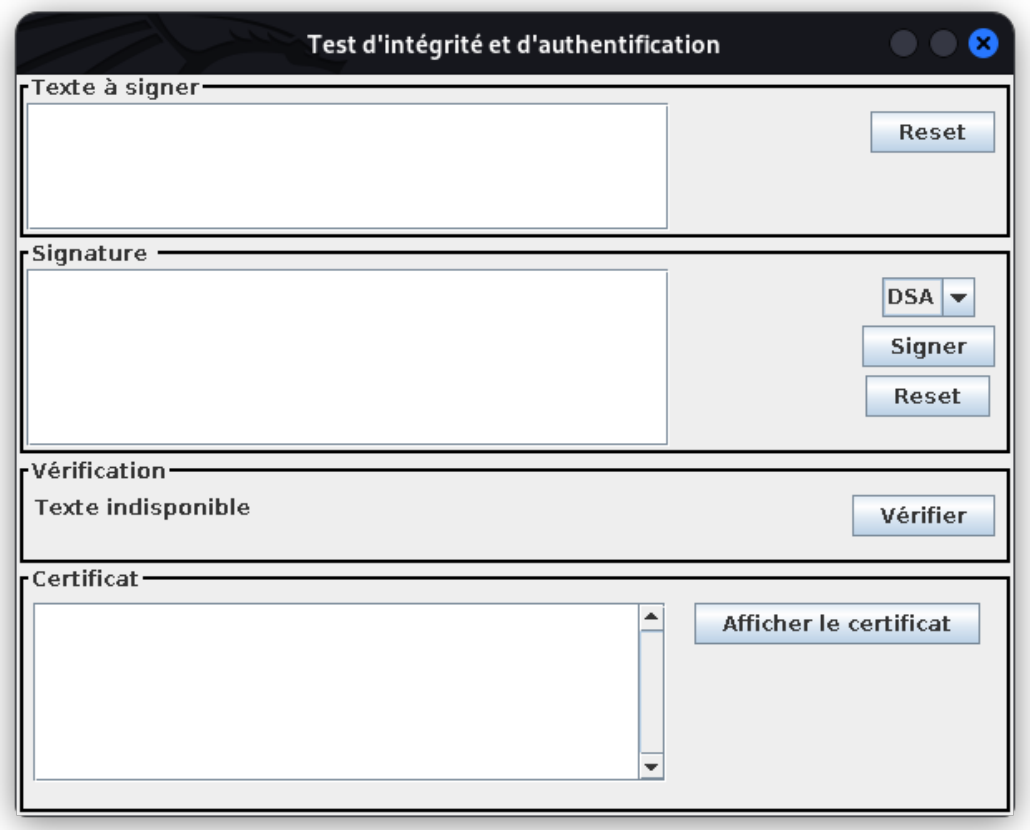

FIGURE 1 – L'interface graphique

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 :

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

```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

"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

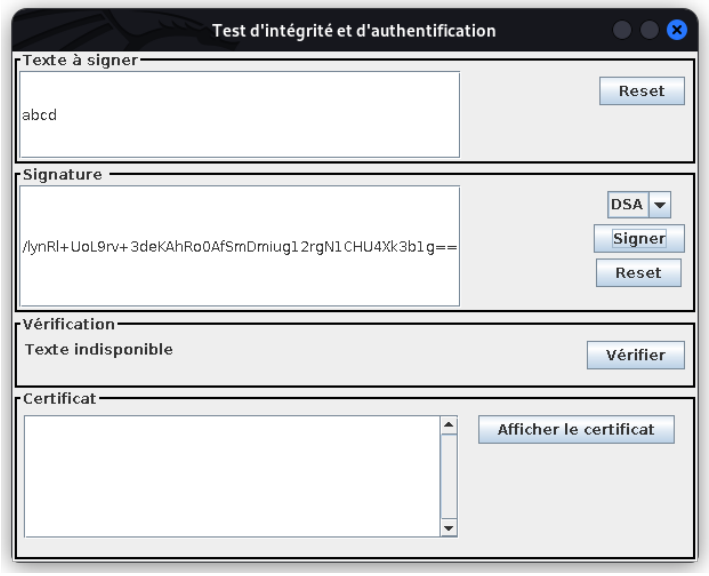

FIGURE 7 – Exemple signature DSA

Voici le résultat du même exemple mais en utilisant RSA

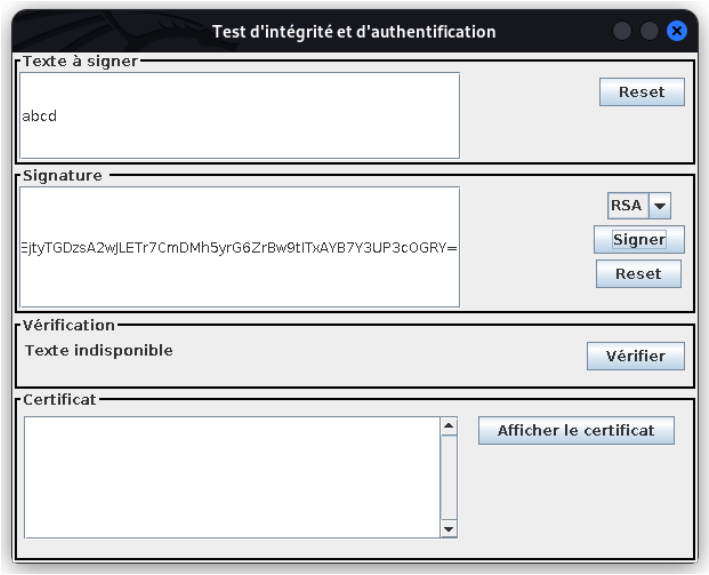

FIGURE 8 – Exemple signature RSA

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 RSA

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

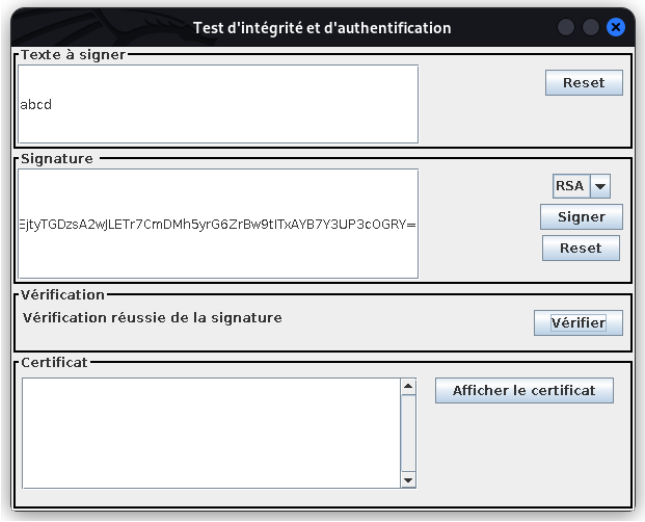

FIGURE 12 – Signature valide

Voici un exemple d'échec de vérification de signature

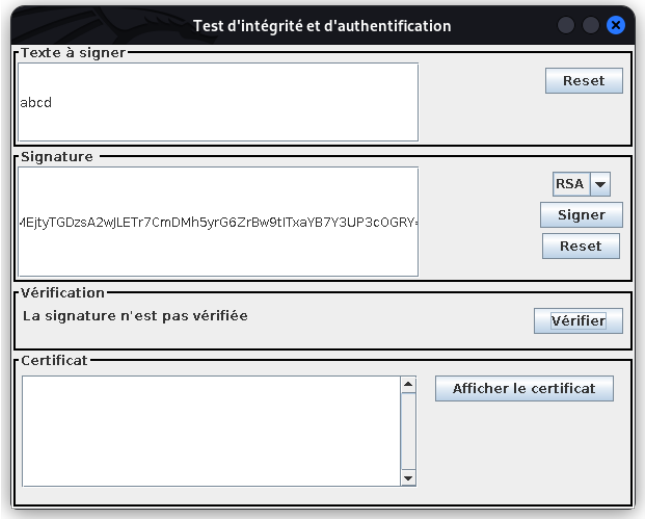

FIGURE 13 – Echec de validation de signature

5.3 Génération des certificats numériques

L'interface offre à l'utilisateur la possibilité de générer des certificat numériques en se basant sur l'algorithme choisi par l'utilisateur (RSA ou DSA) et en tenant compte des clé publics et privées générés par notre programme, comme nous avons vu dans la partie de génération de signature.

Pour générer le certificat de l'algorithme DSA, nous avons utilisés la fonction suivantes qui prend en entrée et qui génère en sortie, cette fonction se base sur les certificats numérique

X.509, et il retourne en sortie le certificat DSA, notons que la fonction de hachage que nous avons choisi est **SHA256**

```
public static String generateCertificate_DSA(PrivateKey privateKey, PublicKey publicKey) throws NoSuchAlgorithmException, IOException, OperatorCreationException, CertificateException {
    X500Name issuer = new X500Name("CN=Autorité de certification");
    BigInteger serialNumber = BigInteger.valueOf(System.currentTimeMillis());
    Date notBefore = new Date(System.currentTimeMillis());
    Date notAfter = new Date(System.currentTimeMillis() + 365 * 24 * 60 * 60 * 1000L);
    X500Name subject = new X500Name("CN=Certificat");
    JcaX509v3CertificateBuilder certBuilder = new JcaX509v3CertificateBuilder(issuer, serialNumber, notBefore, notAfter, subject, publicKey);
    ContentSigner signer = new JcaContentSignerBuilder("SHA256WithDSA").build(privateKey);
    X509CertificateHolder certHolder = certBuilder.build(signer);
    X509Certificate cert = new JcaX509CertificateConverter().getCertificate(certHolder);

    StringWriter stringWriter = new StringWriter();
    PemWriter pemWriter = new PemWriter(stringWriter);
    pemWriter.writeObject(new PemObject("CERTIFICATE", cert.getEncoded()));
    pemWriter.flush();
    pemWriter.close();

    return stringWriter.toString();
}
```

FIGURE 14 – Fonction qui génère certificat numérique DSA

Pour ce qui concerne la fonction qui génère les certificats numériques RSA, la fonction a la même syntaxe que celle du DSA, la figure suivante présente le code de cette fonction.

```
public static String generateCertificate_RSA(PrivateKey privateKey, PublicKey publicKey) throws NoSuchAlgorithmException, IOException, OperatorCreationException, CertificateException {
    X500Name issuer = new X500Name("CN=Autorité de certification");
    BigInteger serialNumber = BigInteger.valueOf(System.currentTimeMillis());
    Date notBefore = new Date(System.currentTimeMillis());
    Date notAfter = new Date(System.currentTimeMillis() + 365 * 24 * 60 * 60 * 1000L);
    X500Name subject = new X500Name("CN=My Cert");
    JcaX509v3CertificateBuilder certBuilder = new JcaX509v3CertificateBuilder(issuer, serialNumber, notBefore, notAfter, subject, publicKey);
    ContentSigner signer = new JcaContentSignerBuilder("SHA256WithRSA").build(privateKey);
    X509CertificateHolder certHolder = certBuilder.build(signer);
    X509Certificate cert = new JcaX509CertificateConverter().getCertificate(certHolder);

    StringWriter stringWriter = new StringWriter();
    PemWriter pemWriter = new PemWriter(stringWriter);
    pemWriter.writeObject(new PemObject("CERTIFICATE", cert.getEncoded()));
    pemWriter.flush();
    pemWriter.close();

    return stringWriter.toString();
}
```

FIGURE 15 – Fonction qui génère certificat numérique RSA

Pour ce qui concerne le bouton qui génère le certificat, nous avons utilisé le code présenté dans la figure suivante qui permet qu'à chaque fois ce bouton est cliqué, le programme vérifie l'algorithme de signature utilisé et puis générer le certificat qui correspond a cet algorithme, puis l'afficher dans la partie qui concerne le certificat.

```
certificat.addActionListener(new ActionListener(){
 public void actionPerformed(ActionEvent evt) {
   String Certificat;
   if (signature.getSelectedItem().toString().equalsIgnoreCase("DSA")) {
     try {
       Certificat = generateCertificate_DSA(keyPair_DSA.getPrivate(),keyPair_DSA.getPublic());
       Certificat_Text.setText(Certificat);
     }
     catch(Exception e){

     }
   }
   else {
     try {
       Certificat = generateCertificate_RSA(keyPair_RSA.getPrivate(),keyPair_RSA.getPublic());
       Certificat_Text.setText(Certificat);
     }
     catch(Exception e){

     }
   }
 }
});
```

FIGURE 16 – Bouton qui génère les certificats numériques

Voici un exemple de génération de certificat numérique en utilisant l'algorithme RSA

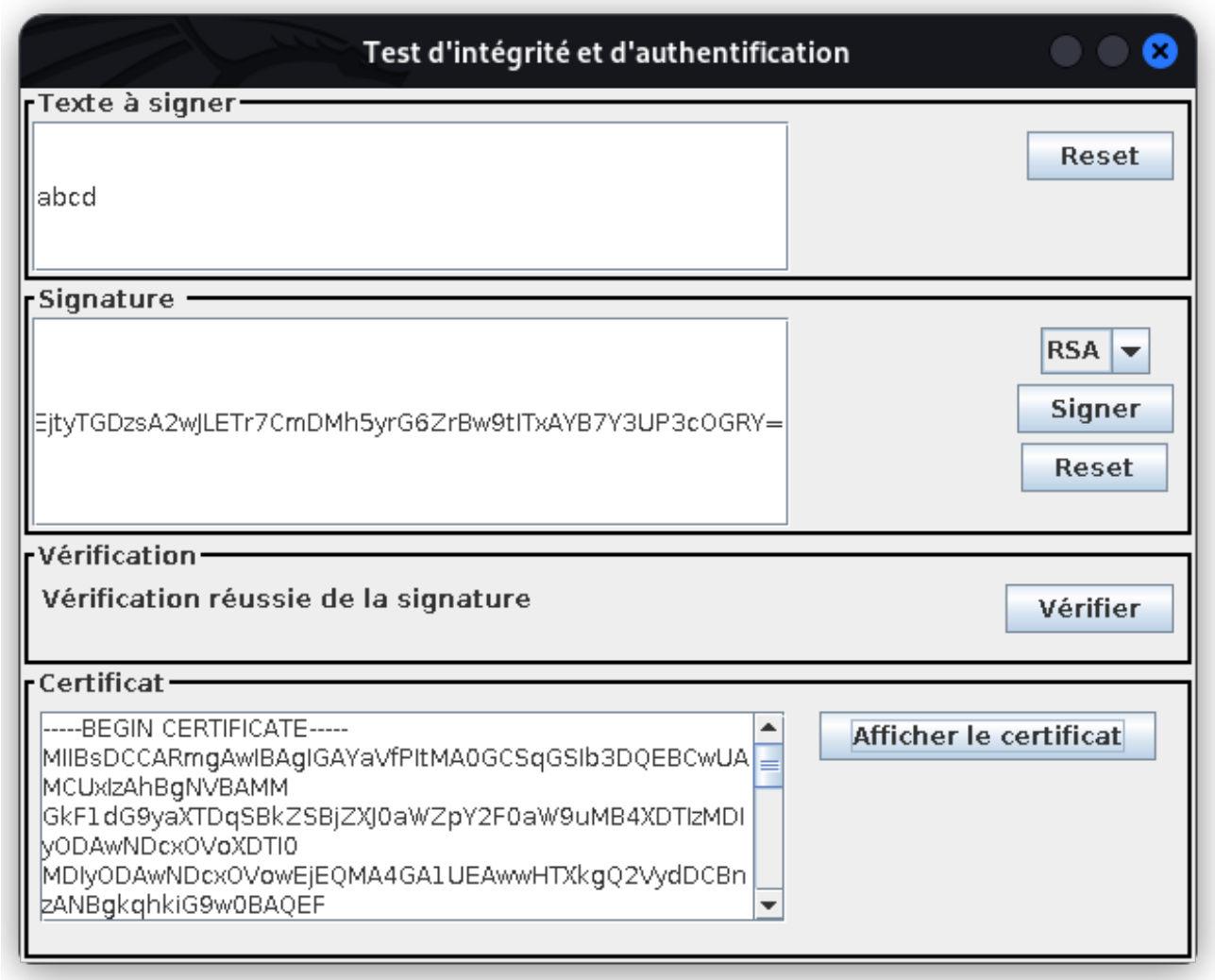

FIGURE 17 – Certificat numérique RSA

5.4 Boutons de réinitialisation

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

Notons que l'éxecution de ce programme nécessite d'avoir le package **org.bouncycastle**, pour cela vous pouver le télecharger en utilisant le lien suivant :
https ://www.bouncycastle.org/latest_releases.html