Comenzado el	viernes, 17 de diciembre de 2021, 17:56
Estado	Finalizado
Finalizado en	viernes, 17 de diciembre de 2021, 18:49
Tiempo empleado	53 minutos 31 segundos
Calificación	**8,72** de 10,00 (**87**%)

1

Puntúa 0,50 sobre 0,50

Correcta

Trieu quina comanda hauríeu de fer servir per signar l'aplicació /home/jars/EAC3P2.jar, fent servir l'àlies appSigner del magatzem de claus ioc.jks.

Seleccione una:

- a. jarsigner -keypass ioc.jks -signedjar /home/jars/EAC3P2-signed.jar /home/jars/EAC3P2.jar appSigner
- b. Cap de les opcions és correcta
- c. jarsigner -keystore ioc.jks -signedjar /home/jars/EAC3P2-signed.jar /home/jars/EAC3P2.jar appSigner ✔
- d. jarsigner -keypass ioc.jks -signedjar /home/jars/EAC3P2.jar appSigner /home/jars/EAC3P2-signed.jar

La resposta és correcta.

La respuesta correcta es: jarsigner -keystore ioc.jks -signedjar /home/jars/EAC3P2-signed.jar /home/jars/EAC3P2.jar appSigner

2

Puntúa -0,13 sobre 0,50

Incorrecta

Trieu quina comanda hauríeu de fer servir per verificar l'aplicació l'autenticitat del codi /home/jars/EAC3P2.jar, fent servir l'alies appSigner del magatzem de claus ioc.jks.

Seleccione una:

- a. jarsigner -verbose -certs -verify EAC3P2-signed.jar
- b. jarsigner -verbose -certify -verified EAC3P2-signed.jar
- c. jarverifier -verbose -certs -verify EAC3P2-signed.jar
- d. Cap de les opcions és correcta ✖

La resposta és incorrecta.

La respuesta correcta es: jarsigner -verbose -certs -verify EAC3P2-signed.jar

3

Puntúa 1,40 sobre 1,50

Parcialmente correcta

Quan s'inicia una connexió segura mitjançant SSL, les dues parts implicades (client i servidor) han de dur a terme prèviament un procés de negociació, o *handshake* en anglès, en el qual s'acorden els paràmetres i la informació criptogràfica que caldrà usar (bàsicament, les claus de xifrat o firma).

Tot seguit es descriuen les etapes d'aquest procés, indicant breument quines tasques es duen a terme en cada cas. El nom de cada etapa es precedeix pel nom de l'entitat qui l'executa (client o servei) i si és opcional o no.

Ompliu les següents descripcions breus amb la seva etapa corresponent:

[CLIENT] Hello: ✔ el client indica que vol posar en marxa una connexió segura SSL. Per fer-ho, indica fins i tot quina versió d'SSL, conjunt d'algorismes criptogràfics i mides de clau suporta.

[SERVEI] Hello: ✔ el servei accepta la petició i tria la darrera versió possible entre les suportades pel client de cara fer ús el protocol segur. Llavors, escull l'algorisme i mida de clau més segur entre els disponibles. Un cop fetes aquestes tries, les comunica al client.

[SERVEI, opcional] Enviament certificat: ✔ aquesta etapa només es realitza si s'usa SSL per fer autenticació simple de servei, però cal dir que aquest és un cas molt habitual quan s'usa SSL. El servei envia el seu certificat digital al client, que conté la seva clau pública i la seva identitat.

[SERVEI, opcional] Petició de certificat: ✔ en cas de fer servir SSL per fer autenticació mútua amb el client, el servei li demana el seu certificat digital.

[CLIENT, opcional] Intercanvi de clau: ✘ aquesta etapa només s'executa per alguns algorismes criptogràfics, que requereixen informació extra, a part d'una clau pública, per establir un canal de comunicacions segur. El servei remet al client aquesta informació addicional.

[SERVEI] Hello finalitzat: ✔ el servei passa el testimoni al client de cara a continuar amb el procés de negociació del protocol.

[CLIENT, opcional] Enviament de certificat: ✔ en cas que el servei demani el certificat del client a l'etapa 4, el client tramet el seu certificat digital. Cal dir que aquest és un cas menys freqüent quan s'usa SSL.

[CLIENT, opcional] Intercanvi de clau: ✔ aquesta etapa és idèntica a l'etapa 5, però resolta per la part del client.

[CLIENT, opcional] Validació certificat del client: ✔ en el cas que es requereixi autenticació del client (veure etapes 4 i 7), s'envia al servei un missatge signat digitalment amb la clau privada del client. Això serveix per demostrar que el client realment disposa de la clau privada associada al certificat que ha tramès, i no ha remès el d'una altra entitat.

[CLIENT] Canvi a mode xifrat: ✔ el client avisa al servei que passa a mode xifrat de comunicacions. Fins ara, totes les comunicacions eren en clar.

[CLIENT] Senyal de fi: ✔ el client indica que la seva part de la negociació ha finalitzat i està llest per iniciar les

comunicacions en mode segur. Aquesta etapa marca el final de la negociació SSL.

[SERVEI] Canvi a mode xifrat: ✔ el servei avisa que ell també passa a mode xifrat de comunicacions.

[SERVEI] Senyal de fi: ✔ el servei indica que, per la seva banda, també ha finalitzat la negociació. Aquesta etapa marca el final del *handshake* SSL.

[CLIENT-SERVEI] Intercanvi de dades: ✔ un cop finalitzada la negociació, el client i el servei porten a terme l'intercanvi de dades segur usant els paràmetres escollits a la negociació, fruit de les etapes anteriors.

[CLIENT-SERVEI] Tancament de connexió: ✔ un cop l'intercanvi de dades finalitza, es tanca la connexió a totes dues bandes.

La resposta és parcialment correcta.

Ha seleccionado correctamente 14.
La respuesta correcta es:

Quan s'inicia una connexió segura mitjançant SSL, les dues parts implicades (client i servidor) han de dur a terme prèviament un procés de negociació, o *handshake* en anglès, en el qual s'acorden els paràmetres i la informació criptogràfica que caldrà usar (bàsicament, les claus de xifrat o firma).

Tot seguit es descriuen les etapes d'aquest procés, indicant breument quines tasques es duen a terme en cada cas. El nom de cada etapa es precedeix pel nom de l'entitat qui l'executa (client o servei) i si és opcional o no.

Ompliu les següents descripcions breus amb la seva etapa corresponent:

[[CLIENT] Hello:] el client indica que vol posar en marxa una connexió segura SSL. Per fer-ho, indica fins i tot quina versió d'SSL, conjunt d'algorismes criptogràfics i mides de clau suporta.

[[SERVEI] Hello:] el servei accepta la petició i tria la darrera versió possible entre les suportades pel client de cara fer ús el protocol segur. Llavors, escull l'algorisme i mida de clau més segur entre els disponibles. Un cop fetes aquestes tries, les comunica al client.

[[SERVEI, opcional] Enviament certificat:] aquesta etapa només es realitza si s'usa SSL per fer autenticació simple de servei, però cal dir que aquest és un cas molt habitual quan s'usa SSL. El servei envia el seu certificat digital al client, que conté la seva clau pública i la seva identitat.

[[SERVEI, opcional] Petició de certificat:] en cas de fer servir SSL per fer autenticació mútua amb el client, el servei li demana el seu certificat digital.

[[SERVEI, opcional] Intercanvi de clau:] aquesta etapa només s'executa per alguns algorismes criptogràfics, que requereixen informació extra, a part d'una clau pública, per establir un canal de comunicacions segur. El servei remet al client aquesta informació addicional.

[[SERVEI] Hello finalitzat:] el servei passa el testimoni al client de cara a continuar amb el procés de negociació del protocol.

[[CLIENT, opcional] Enviament de certificat:] en cas que el servei demani el certificat del client a l'etapa 4, el client tramet el seu certificat digital. Cal dir que aquest és un cas menys freqüent quan s'usa SSL.

[[CLIENT, opcional] Intercanvi de clau:] aquesta etapa és idèntica a l'etapa 5, però resolta per la part del client.

[[CLIENT, opcional] Validació certificat del client:] en el cas que es requereixi autenticació del client (veure etapes 4 i 7), s'envia al servei un missatge signat digitalment amb la clau privada del client. Això serveix per demostrar que el client realment disposa de la clau privada associada al certificat que ha tramès, i no ha remès el d'una altra entitat.

[[CLIENT] Canvi a mode xifrat:] el client avisa al servei que passa a mode xifrat de comunicacions. Fins ara, totes les comunicacions eren en clar.

[[CLIENT] Senyal de fi:] el client indica que la seva part de la negociació ha finalitzat i està llest per iniciar les comunicacions en mode segur. Aquesta etapa marca el final de la negociació SSL.

[[SERVEI] Canvi a mode xifrat:] el servei avisa que ell també passa a mode xifrat de comunicacions.

[[SERVEI] Senyal de fi:] el servei indica que, per la seva banda, també ha finalitzat la negociació. Aquesta etapa marca el final del *handshake* SSL.

[[CLIENT-SERVEI] Intercanvi de dades:] un cop finalitzada la negociació, el client i el servei porten a terme l'intercanvi de dades segur usant els paràmetres escollits a la negociació, fruit de les etapes anteriors.

[[CLIENT-SERVEI] Tancament de connexió:] un cop l'intercanvi de dades finalitza, es tanca la connexió a totes dues bandes.

4

Puntúa 2,50 sobre 2,50

Correcta

El següent codi mostra un exemple de gestor de confiança, pensat només per una aplicació client. Aquest carrega els certificats d'emissors de confiança des d'un magatzem de claus i els usa per validar el certificat d'un servidor. Si el certificat és correcte, mostra per pantalla el del seu emissor. Ompliu els buits existents al codi amb les opcions que teniu

```
public class MyTrustManager implements X509TrustManager {

 private ArrayList(X509Certificate) certs = new ArrayList(X509Certificate)();

 public MyTrustManager() throws Exception {

   keyStore.load(new FileInputStream("SSL/ClientKeyStore.jks"), "client".toCharArray());
   Enumeration<String> aliases = keyStore.aliases();
   while(aliases.hasMoreElements()) {
     String a = aliases.nextElement();
     if (keyStore.isCertificateEntry(a)) {
       certs.add((X509Certificate)keyStore.getCertificate(a));
     }

   }
}

@Override
   public void checkClientTrusted(X509Certificate[] chain, String authType)
               throws CertificateException {
     throw new CertificateException("Aquest gestor només és per aplicacions client.");
   }

@Override
   public void checkServerTrusted(X509Certificate[] chain, String authType)
               throws CertificateException {
       try {
         Validator val = Validator.getInstance(Validator.TYPE_PKIX, Validator.VAR_TLS_CLIENT, certs);
         X509Certificate[] result = val.validate(chain);
         for (X509Certificate c: result) {
           System.out.println("El següent certificat de confiança valida el servei:");
           System.out.println(c);
         }
       } catch (Exception e) {
         throw new CertificateException("El certificat del servei no és de confiança.");
       }
   }

 @Override
   public X509Certificate[] getAcceptedIssuers() {
     X509Certificate[] issuers = new X509Certificate[certs.size()];
     for (int i = 0; i< issuers.length; i++){
       issuers[i] = certs.get(i);
     }
     return issuers;
   }
}
```

La resposta és correcta.

La respuesta correcta es:
El següent codi mostra un exemple de gestor de confiança, pensat només per una aplicació client. Aquest carrega els certificats d'emissors de confiança des d'un magatzem de claus i els usa per validar el certificat d'un servidor. Si el certificat és correcte, mostra per pantalla el del seu emissor. Ompliu els buits existents al codi amb les opcions que teniu

```
public class MyTrustManager implements X509TrustManager {

  private ArrayList(X509Certificate) certs = [new ArrayList(X509Certificate)();]

  public MyTrustManager() throws Exception {

    keyStore.load(new FileInputStream("SSL/ClientKeyStore.jks"), "client".toCharArray());
    Enumeration<String> aliases = keyStore.aliases();
    while(aliases.hasMoreElements()) {
      String a = aliases.nextElement();
      if (keyStore.isCertificateEntry(a)) {
        [certs.add((X509Certificate)keyStore.getCertificate(a)); ]
      }

    }
}

 @Override
    public void checkClientTrusted(X509Certificate[] chain, String authType)
                throws CertificateException {
      throw new CertificateException("Aquest gestor només és per aplicacions client.");
    }

 @Override
    public void checkServerTrusted(X509Certificate[] chain, String authType)
                throws CertificateException {
        try {
          Validator val = [Validator.getInstance(Validator.TYPE_PKIX, Validator.VAR_TLS_CLIENT, certs);]
          X509Certificate[] result = [val.validate(chain);]
          for (X509Certificate c: result) {
            System.out.println("El següent certificat de confiança valida el servei:");
            System.out.println(c);
          }
        } catch (Exception e) {
          throw new CertificateException("El certificat del servei no és de confiança.");
        }
    }

  @Override
    public X509Certificate[] getAcceptedIssuers() {
      [X509Certificate[] issuers = new X509Certificate[certs.size()];]
      for (int i = 0; i< issuers.length; i++){
        [issuers[i] = certs.get(i);]
      }
      [return issuers;]
    }
}
```

5

Puntúa 2,50 sobre 2,50

Correcta

Aquest exercici centra en l'algorisme anomenat AES (*Advanced Encryption Standrad*), que es considera més segur i l'estàndard en algorismes de xifrat des de l'any 2001. Ompliu els buits que hi ha al codi amb alguna de les opcions disponibles.

```
import java.security.Key;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.spec.SecretKeySpec;
/**
 * Exemple de xifrat i desxifrat amb l'algorisme AES.
 *
 */
public class AESSymetricCrypto {
   public static void main(String[] args) throws Exception {
   // Generem una clau de 128 bits per AES
```

KeyGenerator keyGenerator = KeyGenerator.getInstance("AES"); ✔

```
   keyGenerator.init(128);
```

Key key = keyGenerator.generateKey(); ✔

```
   // Alternativament, una clau que volem que tingui almenys 16 bytes
   // i ens quedem amb els bytes 0 a 15
   key = new SecretKeySpec("una clau de 16 bytes".getBytes(),  0, 16, "AES");

   // Text a encriptar
   String text = "Aquest és el text que volem xifrar";
   // S'obté un xifrador AES
```

Cipher aes = Cipher.getInstance("AES/ECB/PKCS5Padding"); ✔

```
   // S'inicialitza pel xifratge i s'encripta el text que hem de passar a bytes.
```

aes.init(Cipher.ENCRYPT_MODE, key); ✔

byte[] xifrat = aes.doFinal(text.getBytes()); ✔

```
      // S'escriu byte a byte en hexadecial el text
      for (byte b : xifrat) {
         System.out.print(Integer.toHexString(0xFF & b));
      }
      System.out.println();
      // Es desxifra amb la mateixa clau
```

aes.init(Cipher.DECRYPT_MODE, key); ✔

byte[] desxifrat = aes.doFinal(xifrat); ✔

```
      // Text obtingut, ha de ser igual a l'original
      System.out.println(new String(desxifrat));
   }
}
```

La resposta és correcta.

La respuesta correcta es:
Aquest exercici centra en l'algorisme anomenat AES (*Advanced Encryption Standrad*), que es considera més segur i l'estàndard en algorismes de xifrat des de l'any 2001. Ompliu els buits que hi ha al codi amb alguna de les opcions disponibles.

```
import java.security.Key;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.spec.SecretKeySpec;
/**
 * Exemple de xifrat i desxifrat amb l'algorisme AES.
 *
 */
public class AESSymetricCrypto {
   public static void main(String[] args) throws Exception {
      // Generem una clau de 128 bits per AES
      [KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");]
      keyGenerator.init(128);
      [Key key = keyGenerator.generateKey();]

      // Alternativament, una clau que volem que tingui almenys 16 bytes
      // i ens quedem amb els bytes 0 a 15
      key = new SecretKeySpec("una clau de 16 bytes".getBytes(),  0, 16, "AES");

      // Text a encriptar
      String text = "Aquest és el text que volem xifrar";
      // S'obté un xifrador AES
      [Cipher aes = Cipher.getInstance("AES/ECB/PKCS5Padding");]
      // S'inicialitza pel xifratge i s'encripta el text que hem de passar a bytes.

      [aes.init(Cipher.ENCRYPT_MODE, key);]
      [byte[] xifrat = aes.doFinal(text.getBytes());]
      // S'escriu byte a byte en hexadecial el text
      for (byte b : xifrat) {
         System.out.print(Integer.toHexString(0xFF & b));
      }
      System.out.println();
      // Es desxifra amb la mateixa clau

      [aes.init(Cipher.DECRYPT_MODE, key);]
      [byte[] desxifrat = aes.doFinal(xifrat);]
      // Text obtingut, ha de ser igual a l'original
      System.out.println(new String(desxifrat));
   }
}
```

6

Puntúa 1,94 sobre 2,50

Parcialmente correcta

El sistema per xifrar i desxifrar mitjançant algorismes de clau pública que ofereix JCE és idèntic a l'emprat per clau simètrica. També es basa en la mateixa classe `Cipher`. Per xifrar, cal usar la clau pública com a paràmetre i, al desxifrar, la clau privada.

El següent codi serveix per xifrar i desxifrar una cadena text utilitzant l'Algorisme RSA. Ompliu els buits que hi ha al codi amb les opcions disponibles.

```
public class ExempleRSA {
   private static Cipher rsa;
   public static void main(String[] args) throws Exception {
      // Generar el parell de claus
      KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
      KeyPair keyPair = keyPairGenerator.generateKeyPair();
      PublicKey publicKey = keyPair.getPublic();
      PrivateKey privateKey = keyPair.getPrivate();
      // Es guarda i es recupera el fitxer la clau publica
      saveKey(publicKey, "publickey.dat");
      publicKey = loadPublicKey("publickey.dat");
      // Es guarda i es recupera el fitxer la clau privada
      saveKey(privateKey, "privatekey.dat");
      privateKey = loadPrivateKey("privatekey.dat");
      // Obtenir la classe per encriptar/desencriptar
      keyFactory.generatePrivate(keySpec);
      // Text a encriptar
      String text = "Text que volem encriptar";
      // Es xifra
      rsa.init(Cipher.ENCRYPT_MODE, publicKey);
      byte[] xifrat = rsa.doFinal(text.getBytes());
      // S'escriu el xifrat per veureu
      for (byte b : xifrat) {
         System.out.print(Integer.toHexString(0xFF & b));
      }
      System.out.println();
      // Es desxifra
      rsa.init(Cipher.DECRYPT_MODE, privateKey);
      byte[] bytesDesxifrats = rsa.doFinal(xifrat);
      String textXifrat= new String(bytesDesxifrats);
      // S'escriu el text desxifrat
      System.out.println(textXifrat);
   }
   private static PublicKey loadPublicKey(String fileName) throws Exception {
      FileInputStream fis = new FileInputStream(fileName);
      int numBtyes = fis.available();
      byte[] bytes = new byte[numBtyes];
      fis.read(bytes);
      fis.close();
      KeyFactory keyFactory = KeyFactory.getInstance("RSA");
      KeySpec keySpec = new X509EncodedKeySpec(bytes);
      PublicKey keyFromBytes = keyPair.getPublic();
      return keyFromBytes;
   }
   private static PrivateKey loadPrivateKey(String fileName) throws Exception {
      FileInputStream fis = new FileInputStream(fileName);
      int numBtyes = fis.available();
      byte[] bytes = new byte[numBtyes];
      fis.read(bytes);
```

```
      fis.close();
      KeyFactory keyFactory = KeyFactory.getInstance("RSA");
      KeySpec keySpec = new PKCS8EncodedKeySpec(bytes);
      PrivateKey keyFromBytes = keyFactory.generatePrivate(keySpec);
      return keyFromBytes;
   }
   private static void saveKey(Key key, String fileName) throws Exception {
      byte[] publicKeyBytes = key.getEncoded();
      FileOutputStream fos = new FileOutputStream(fileName);
      fos.write(publicKeyBytes);
      fos.close();
   }
}
```

La resposta és parcialment correcta.
Ha seleccionado correctamente 7.
La respuesta correcta es:

El sistema per xifrar i desxifrar mitjançant algorismes de clau pública que ofereix JCE és idèntic a l'emprat per clau simètrica. També es basa en la mateixa classe `Cipher`. Per xifrar, cal usar la clau pública com a paràmetre i, al desxifrar, la clau privada.

El següent codi serveix per xifrar i desxifrar una cadena text utilitzant l'Algorisme RSA. Ompliu els buits que hi ha al codi amb les opcions disponibles.

```
public class ExempleRSA {
   private static Cipher rsa;
   public static void main(String[] args) throws Exception {
      // Generar el parell de claus
      [KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");]
      KeyPair keyPair = keyPairGenerator.generateKeyPair();
      PublicKey publicKey = keyPair.getPublic();
      PrivateKey privateKey = [keyPair.getPrivate();]
      // Es guarda i es recupera el fitxer la clau publica
      saveKey(publicKey, "publickey.dat");
      publicKey = loadPublicKey("publickey.dat");
      // Es guarda i es recupera el fitxer la clau privada
      saveKey(privateKey, "privatekey.dat");
      privateKey = loadPrivateKey("privatekey.dat");
      // Obtenir la classe per encriptar/desencriptar
      [rsa = Cipher.getInstance("RSA/ECB/PKCS1Padding");]
      // Text a encriptar
      String text = "Text que volem encriptar";
      // Es xifra
      [rsa.init(Cipher.ENCRYPT_MODE, publicKey);]
      [byte[] xifrat = rsa.doFinal(text.getBytes());]
      // S'escriu el xifrat per veureu
      for (byte b : xifrat) {
         System.out.print(Integer.toHexString(0xFF & b));
      }
      System.out.println();
      // Es desxifra
      [rsa.init(Cipher.DECRYPT_MODE, privateKey);]
      [byte[] bytesDesxifrats = rsa.doFinal(xifrat);]
      String textXifrat= new String(bytesDesxifrats);
      // S'escriu el text desxifrat
      System.out.println(textXifrat);
   }
   private static PublicKey loadPublicKey(String fileName) throws Exception {
      FileInputStream fis = new FileInputStream(fileName);
      int numBtyes = fis.available();
      byte[] bytes = new byte[numBtyes];
      fis.read(bytes);
      fis.close();
      KeyFactory keyFactory = KeyFactory.getInstance("RSA");
      KeySpec keySpec = new X509EncodedKeySpec(bytes);
      PublicKey keyFromBytes = [keyFactory.generatePublic(keySpec);]
      return keyFromBytes;
   }
   private static PrivateKey loadPrivateKey(String fileName) throws Exception {
      FileInputStream fis = new FileInputStream(fileName);
      int numBtyes = fis.available();
      byte[] bytes = new byte[numBtyes];
      fis.read(bytes);
      fis.close();
      KeyFactory keyFactory = KeyFactory.getInstance("RSA");
      KeySpec keySpec = new PKCS8EncodedKeySpec(bytes);
      PrivateKey keyFromBytes = keyFactory.generatePrivate(keySpec);
      return keyFromBytes;
   }
   private static void saveKey(Key key, String fileName) throws Exception {
      byte[] publicKeyBytes = key.getEncoded();
      FileOutputStream fos = new FileOutputStream(fileName);
      [fos.write(publicKeyBytes);]
      fos.close();
   }
}
```

◄ ENUNCIAT EAC3P2

Ir a...

M9 LLIURAMENT EAC3P2 ►