Software Security – Lab Session

Brandon Fontany-Legall

brandon.fontany–legall@etu.unice.fr

Le code utilisant la méthode readAllBytes() de StringBuilder, il est nécessaire d'utiliser une version de Java supérieure ou égale à Java 9

3DES

1 DATA ENCRYPTION STANDARD

What is the size of DES cipher key's size ?

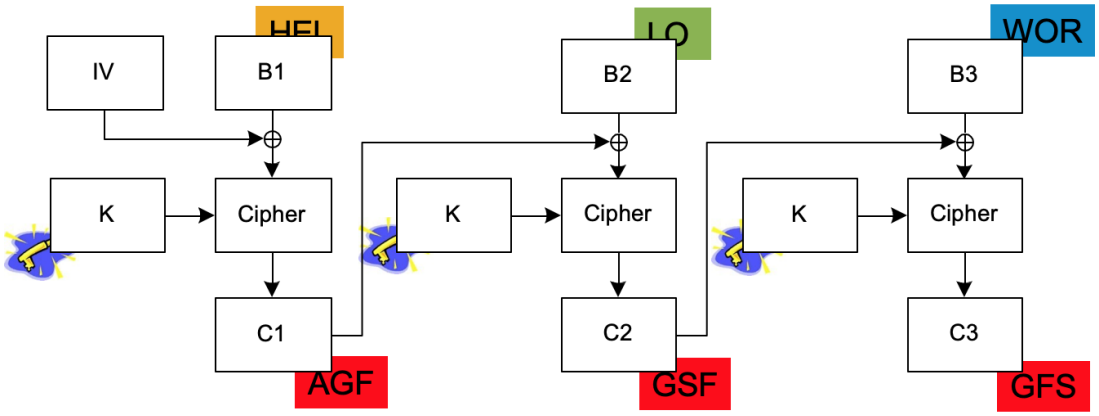
La taille d'une clef DES cipher est de 64 bits dont 56 sont pour le payload et 8 pour les bits de parité.

What are the size of the cipher blocks ?

La taille d'un block cipher est de 64 bits (8 bytes)

2 DES/CBC/NOPADDING

Explain CBC with diagram



Le mode CBC (Cipher Block Chaining) est un mode dans lequel le chiffrement de chaque bloc dépend du chiffrement précédent. En effet, sur chaque block est appliqué un OU exclusif avec le chiffrement du bloc précédent pour ensuite le chiffré. Pour le cas du premier block le OU exclusif se fait avec un vecteur d'initialisation (IV) qui est généré de manière aléatoire.

Explain NoPadding

Le padding est un remplissage des blocks de manière à que chaque block soit de taille égale et en accord avec l'algorithme de chiffrement. En effet, beaucoup d'algorithm de de chiffrement utilisent des blocks de taille fix, il est donc important que la taille des block en entrée soit de même taille que ceux qu'utilise l'algorithme. Dans notre cas, le **NoPadding** ne fait pas ce remplissage, il faut donc faire attention à ce que nos blocks soit de bonne taille.

Perform 3DES in EBC mode encryption in TripleDES ::private Vector encryptEBC(...)

```
1 private Vector encryptECB(FileInputStream in,
2                            FileOutputStream out,
```

```
                            String KeyGeneratorInstanceName,
                            String CipherInstanceName){
        try{

            KeyGenerator keyGen = KeyGenerator.getInstance(KeyGeneratorInstanceName);
            keyGen.init(new SecureRandom());
            SecretKey secretLevel1 = keyGen.generateKey();
            SecretKey secretLevel2 = keyGen.generateKey();
            SecretKey secretLevel3 = keyGen.generateKey();

            Cipher cipherLevel1 = Cipher.getInstance(CipherInstanceName);
            cipherLevel1.init(Cipher.ENCRYPT_MODE, secretLevel1);

            Cipher cipherLevel2 = Cipher.getInstance(CipherInstanceName);
            cipherLevel2.init(Cipher.DECRYPT_MODE, secretLevel2);

            Cipher cipherLevel3 = Cipher.getInstance(CipherInstanceName);
            cipherLevel3.init(Cipher.ENCRYPT_MODE, secretLevel3);

            StringBuilder stringBuilder = new StringBuilder();

            byte[] plainText  = in.readAllBytes();
            byte[] firstPass = cipherLevel1.doFinal(plainText);
            byte[] secondPass = cipherLevel2.doFinal(firstPass);
            byte[] thirdPass = cipherLevel3.doFinal(secondPass);

            out.write(thirdPass);
            out.close();

            Vector<SecretKey> vector = new Vector();
            vector.add(secretLevel1);
            vector.add(secretLevel2);
            vector.add(secretLevel3);

            return vector;

        }catch(Exception e){
            e.printStackTrace();
            return null;
        }

    }
```

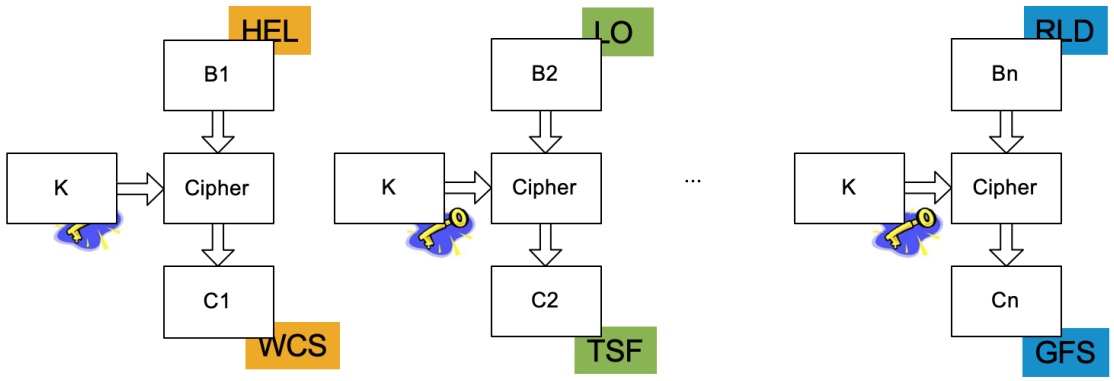
Perform 3DES in EBC mode decryption in TripleDES ::private void decryptEBC(...)

```
private void decryptECB(Vector Parameters,
                        FileInputStream in,
                        FileOutputStream out,
                        String CipherInstanceName){
        try{

            SecretKey secretLevel1 = (SecretKey) Parameters.get(0);
            SecretKey secretLevel2 = (SecretKey) Parameters.get(1);
            SecretKey secretLevel3 = (SecretKey) Parameters.get(2);

            Cipher cipherLevel1 = Cipher.getInstance(CipherInstanceName);
            cipherLevel1.init(Cipher.DECRYPT_MODE, secretLevel3);

            Cipher cipherLevel2 = Cipher.getInstance(CipherInstanceName);
            cipherLevel2.init(Cipher.ENCRYPT_MODE, secretLevel2);
```

```
16
17              Cipher cipherLevel3 = Cipher.getInstance(CipherInstanceName);
18              cipherLevel3.init(Cipher.DECRYPT_MODE, secretLevel1);
19
20              byte[] plainText  = in.readAllBytes();
21              byte[] firstPass = cipherLevel1.doFinal(plainText);
22              byte[] secondPass = cipherLevel2.doFinal(firstPass);
23              byte[] thirdPass = cipherLevel3.doFinal(secondPass);
24
25              out.write(thirdPass);
26              out.close();
27
28          }catch(Exception e){
29              e.printStackTrace();
30          }
31
32      }
```

3 DES/EBC/NOPADDING

Explain EBC and its advantages over CBC with diagram



Le mode de cryptage ECB est un mode beaucoup plus simple que CBC. En effet, le mode ECB va chiffrer les blocks séparément des autres. Chaque block se fait donc chiffré avec la même clef.

Les principaux **avantage** sont :

- Une mise en place plus simple que CBC. En effet, ECB n'a pas besoin de faire d'opération en fonction des blocks différent ni de faire la génération de l'IV.
- La possibilité de chiffré en parallèle. En effet, vu que dans le mode ECB, chaque bloque sont traité indépendamment, il est possible de chiffré les block en parallèle permettant d'aller plus vite.
- La propagation d'erreur est minimisé. En effet, si un block à une erreur lors du processus de chiffrement, les blocks suivant ne seront pas impacté à l'inverse du mode CBC où tout les blocks suivant seront impacté par l'erreur.

Cependant, et c'est un **désavantage**, lors du chiffrement, si deux block sont identique avant chiffrement, il le seront après chiffrement ouvrant possiblement la porte à certains déduction sur la clé K.

Perform 3DES in CBC mode encryption in TripleDES ::private Vector encryptCBC(...)

```
1  private Vector encryptCBC(FileInputStream in,
2                            FileOutputStream out,
```

```
                            String KeyGeneratorInstanceName,
                            String CipherInstanceName){
        try{

            byte[] iv = new byte[128/8];
            new SecureRandom().nextBytes(iv);
            IvParameterSpec ivspec = new IvParameterSpec(iv);

            KeyGenerator keyGen = KeyGenerator.getInstance(KeyGeneratorInstanceName);
            keyGen.init(new SecureRandom());
            SecretKey secretLevel1 = keyGen.generateKey();
            SecretKey secretLevel2 = keyGen.generateKey();
            SecretKey secretLevel3 = keyGen.generateKey();

            IvParameterSpec IVLevel1 = new IvParameterSpec(new byte[8]);
            IvParameterSpec IVLevel2 = new IvParameterSpec(new byte[8]);
            IvParameterSpec IVLevel3 = new IvParameterSpec(new byte[8]);

            Cipher cipherLevel1 = Cipher.getInstance(CipherInstanceName);
            cipherLevel1.init(Cipher.ENCRYPT_MODE, secretLevel1, IVLevel1);

            Cipher cipherLevel2 = Cipher.getInstance(CipherInstanceName);
            cipherLevel2.init(Cipher.DECRYPT_MODE, secretLevel2, IVLevel2);

            Cipher cipherLevel3 = Cipher.getInstance(CipherInstanceName);
            cipherLevel3.init(Cipher.ENCRYPT_MODE, secretLevel3, IVLevel3);

            StringBuilder stringBuilder = new StringBuilder();

            byte[] plainText  = in.readAllBytes();
            byte[] firstPass = cipherLevel1.doFinal(plainText);
            byte[] secondPass = cipherLevel2.doFinal(firstPass);
            byte[] thirdPass = cipherLevel3.doFinal(secondPass);

            out.write(thirdPass);
            out.close();

            Vector vector = new Vector();
            vector.add(secretLevel1);
            vector.add(IVLevel1);
            vector.add(secretLevel2);
            vector.add(IVLevel2);
            vector.add(secretLevel3);
            vector.add(IVLevel3);

            return vector;

        }catch(Exception e){
            e.printStackTrace();
            return null;
        }
    }
```

Perform 3DES in CBC mode decryption in TripleDES ::private void decryptCBC(...)

```
private void decryptCBC(Vector Parameters,
                        FileInputStream in,
                        FileOutputStream out,
                        String CipherInstanceName){
        try{
```

```
            SecretKey secretLevel1 = (SecretKey) Parameters.get(0);
            IvParameterSpec IVLevel1 = (IvParameterSpec) Parameters.get(1);
            SecretKey secretLevel2 = (SecretKey) Parameters.get(2);
            IvParameterSpec IVLevel2 = (IvParameterSpec) Parameters.get(3);
            SecretKey secretLevel3 = (SecretKey) Parameters.get(4);
            IvParameterSpec IVLevel3 = (IvParameterSpec) Parameters.get(5);

            Cipher cipherLevel1 = Cipher.getInstance(CipherInstanceName);
            cipherLevel1.init(Cipher.DECRYPT_MODE, secretLevel3, IVLevel3);

            Cipher cipherLevel2 = Cipher.getInstance(CipherInstanceName);
            cipherLevel2.init(Cipher.ENCRYPT_MODE, secretLevel2, IVLevel2);

            Cipher cipherLevel3 = Cipher.getInstance(CipherInstanceName);
            cipherLevel3.init(Cipher.DECRYPT_MODE, secretLevel1, IVLevel1);

            byte[] plainText  = in.readAllBytes();
            byte[] firstPass = cipherLevel1.doFinal(plainText);
            byte[] secondPass = cipherLevel2.doFinal(firstPass);
            byte[] thirdPass = cipherLevel3.doFinal(secondPass);

            out.write(thirdPass);
            out.close();

        }catch(Exception e){
            e.printStackTrace();
        }

    }
```

RSA Signature Implementation

1 USE OF JAVA SIGNATURE

1.1 Generation of a public/private key pair

```
public Entity(){
        try{
            KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
            keyPairGenerator.initialize(1024);
            KeyPair keyPair = keyPairGenerator.genKeyPair();

            this.thePublicKey = keyPair.getPublic();
            this.thePrivateKey = keyPair.getPrivate();
        }catch(Exception e){
            System.out.println("Signature error");
            e.printStackTrace();
        }
    }
```

1.2 RSA Signature

```
public byte[] sign(byte[] aMessage){

        try{
            Signature signature = Signature.getInstance("SHA1withRSA");
            signature.initSign(this.thePrivateKey);
```

```
            signature.update(aMessage);

            return signature.sign();
        }catch(Exception e){
            System.out.println("Signature error");
            e.printStackTrace();
            return null;
        }

    }
```

1.3 Check signature

```
public boolean checkSignature(byte[] aMessage, byte[] aSignature, PublicKey aPK){
        try{
            Signature signature = Signature.getInstance("SHA1withRSA");
            signature.initVerify(aPK);
            signature.update(aMessage);

            return signature.verify(aSignature);
        }catch(Exception e){
            System.out.println("Verify signature error");
            e.printStackTrace();
            return false;
        }
    }
```

2 IMPLEMENTATION OF YOUR OWN RSA SIGNATURE

2.1 Signature

```
public byte[] mySign(byte[] aMessage){

        try{
            Cipher cipher = Cipher.getInstance("RSA");
            cipher.init(Cipher.ENCRYPT_MODE, this.thePrivateKey);

            MessageDigest messageDigest = MessageDigest.getInstance("SHA1");
            messageDigest.update(aMessage);
            byte[] digest = messageDigest.digest(aMessage);

            return cipher.doFinal(digest);
        }catch(Exception e){
            System.out.println("Signature error");
            e.printStackTrace();
            return null;
        }

    }
```

2.2 Check signature

```
public boolean myCheckSignature(byte[] aMessage, byte[] aSignature, PublicKey aPK){
        try{
            Cipher cipher = Cipher.getInstance("RSA");
            cipher.init(Cipher.DECRYPT_MODE, aPK);

            byte[] digest1 = cipher.doFinal(aSignature);
```

```
            MessageDigest messageDigest = MessageDigest.getInstance("SHA1");
            messageDigest.update(aMessage);

            return digest1.equals(messageDigest.digest());
        }catch(Exception e){
            System.out.println("Verify signature error");
            e.printStackTrace();
            return false;
        }
    }
```

3 RSA CIPHERING

3.1 RSAEncryption

```
public byte[] encrypt(byte[] aMessage, PublicKey aPK){
        try{
            Cipher cipher = Cipher.getInstance("RSA");
            cipher.init(Cipher.ENCRYPT_MODE, aPK);

            return cipher.doFinal(aMessage);
        }catch(Exception e){
            System.out.println("Encryption error");
            e.printStackTrace();
            return null;
        }
    }
```

3.2 RSADecryption

```
public byte[] decrypt(byte[] aMessage){
        try{
            Cipher cipher = Cipher.getInstance("RSA");
            cipher.init(Cipher.DECRYPT_MODE, this.thePrivateKey);

            return cipher.doFinal(aMessage);
        }catch(Exception e){
            System.out.println("Encryption error");
            e.printStackTrace();
            return null;
        }

    }
```

4 SECURE SESSION KEY EXCHANGE

```
static void KeyExchangeProtocol(){
            Entity Alice, Bob;
            try {
            //  Alice sends her public key to Bob.
            //  Bob generate a DES session key.

            KeyGenerator keyGenerator = KeyGenerator.getInstance("DES");
            SecretKey secretKey = keyGenerator.generateKey();

            //  Bob encrypts it with A l i c e s  public key.
                byte[] bobEncrypt = Bob.encrypt(secretKey.getEncoded(), Alice.
                    thePublicKey);
```

```
12
13              //  Alice decrypts the DES key with her private key.
14                  byte[] aliceDecrypt = Alice.decrypt(bobEncrypt);
15
16              //  Alice sends a message to Bob with her session key
17              //  Bob decrypts the message with the session key.
18                  Bob.decrypt(aliceDecrypt);
19
20              } catch (NoSuchAlgorithmException e) {
21                  e.printStackTrace();
22                  System.exit(1);
23              }
24
25          }
```