Домашна задача бр 2
Информациска Безбедност

Во процесот на изработка на KDC сервер избрав да креирам соодветно Repository за KDC и Сервисен слој за истиот.
Истиот процес го креирам и за секој корисник, и двата сервисни слоја се повикуваат од MessageApp која може да регистрира корисник, корисник може да праќа пораки на некој друг корисник.
Сценарија:

Процес на регистрирање на корисник код:

```
public String registerUser(String name, String surname) throws NoSuchAlgorithmException {
    return this.messageService.registerUser(name, surname);
}
```

Влезна точка за регистрирање на корисник, се повикува messageService за регистрирање на корисници во апликацијата.

```
@Override
public String registerUser(String name, String surname) throws NoSuchAlgorithmException {
    User user = this.userService.save(name, surname);
    kdcService.save(user.getId(), user.getMySecretKey());
    return user.getId();
}
```

Во овој дел се креира корисник преку сервисот за корисници, и се зачувува генерираниот рандом клуч за корисникот во DummyRepository кое содржи хеш табела од корисникИд и клуч на корисникот.
Претпоставка KDC серверот и MessageApp се на иста локација.

```
public User(String name,String surname) throws NoSuchAlgorithmException {
    this.id = UUID.randomUUID().toString();
    this.name=name;
    this.surname=surname;
    this.sessionKeys= new HashMap<>();
    this.mySecretKey= Utils.generateRandomSecretKey();


}
public static SecretKey generateRandomSecretKey() throws NoSuchAlgorithmException {
    KeyGenerator keyGen = KeyGenerator.getInstance("AES");
    keyGen.init(128);
    return keyGen.generateKey();
}
```

Соодветен конструкот за инстанца од класата Usep и методот за креирање на рандом клуч за секој корисник кој потоа се користи и за креирање на сесиски клуч.

Процес на праќање на порака:

- Почетна точка

```
public void sendMessage(String userIdFrom, String userIdTo, byte[] message) throws
UserNotFoundException {
    messageService.save(userIdFrom, userIdTo, message);
}
```

Притоа при праќањето на поракате бидејќи се работа за демо на веб апликација се претпоставува дека се користи SSL.

```
@Override
public void save(String userIdFrom, String userIdTo, byte[] message) throws
UserNotFoundException {
    kdcService.createSessionKeyFor(userIdFrom,userIdTo);
    MessageRepository.messages.add(userService.encryptMessage(userIdFrom, userIdTo,
message));
}
```

При повик на сервисот за пораки за зачувување на порака прво се повикува KDC сервисот за да провери дали постои сесиски клуч помеѓу корисници што сакаат да комуницираат и доколку не постои се креира истиот со следниот код.

```
@Override
public void createSessionKeyFor(String userFromId, String userToId) {
    try {
        User from = userService.findById(userFromId);
        User to = userService.findById(userToId);
        if (!from.getSessionKeys().containsKey(to.getId()) && !
to.getSessionKeys().containsKey(from.getId())) {
            String nonce = from.generateNonce();
            SecretKey sessionKey = Utils.generateRandomSecretKey();
            byte[] encryptedKey = AES.encrypt(sessionKey.getEncoded(), from.getMySecretKey());
            byte[] encryptedLifetime = AES.encrypt(Utils.longToByteArray(KDCRepository.miliseconds),
from.getMySecretKey());
            byte[] encryptedNonce = AES.encrypt(nonce.getBytes(), from.getMySecretKey());
            byte[] encryptedIdTo = AES.encrypt(userToId.getBytes(), from.getMySecretKey());
            KDCSendEncryptedFrameFromUser fromUser = new
KDCSendEncryptedFrameFromUser(encryptedKey,
                    encryptedNonce, encryptedLifetime, encryptedIdTo);
            byte[] encryptedKeyTo = AES.encrypt(sessionKey.getEncoded(), to.getMySecretKey());
            byte[] encryptedIdFrom = AES.encrypt(from.getId().getBytes(), to.getMySecretKey());
            byte[] encryptedLifetimeTo =
AES.encrypt(Utils.longToByteArray(KDCRepository.miliseconds), to.getMySecretKey());
            KDCSendEncryptedFrameToUser toUser = new
KDCSendEncryptedFrameToUser(encryptedKeyTo, encryptedIdFrom, encryptedLifetimeTo);
            KDCSendEncryptedFrame frame = new KDCSendEncryptedFrame(fromUser, toUser);
            if (userService.receiveFromKDC(from.getId(), frame)) {
                userService.setSessionKey(from.getId(), to.getId(), frame);
            } else {
                throw new RuntimeException("An Error Occurred");
            }
        }
```

```
    } catch (UserNotFoundException | NoSuchAlgorithmException | IOException e) {
        System.out.println(e.getMessage());
    }

}
```

При што се верифицира дали корисниците постојат во апликацијата, доколку постојат се проверува дали постои размената веќе помеѓу нив сесиски клуч доколку не постои се креира нонсе од корисникот што сака да комуницира со другиот корисник.

```
User from = userService.findById(userFromId);
        User to = userService.findById(userToId);
public String generateNonce(){
    this.nonce=UUID.randomUUID().toString();
    return nonce;
}
```

Избрана ми UUID што се користи за ид но мислам дека постои добра дефиниција за генерирање на рандом ид без притоа да постојат повторувања.
Потоа се генерира рандом сесиску клуч.

```
SecretKey sessionKey = Utils.generateRandomSecretKey();
```

Се енкриптираат потребните информации за верификација од страна на иницијатор на комуникацијата кои се енкапсулираат во посебен објект KDCEncryptedFrame.
Кој се состои од KDCEncryptedFromUser и KDCEncryptedToUser.

```
 byte[] encryptedKey = AES.encrypt(sessionKey.getEncoded(), from.getMySecretKey());
          byte[] encryptedLifetime = AES.encrypt(Utils.longToByteArray(KDCRepository.miliseconds),
from.getMySecretKey());
          byte[] encryptedNonce = AES.encrypt(nonce.getBytes(), from.getMySecretKey());
          byte[] encryptedIdTo = AES.encrypt(userToId.getBytes(), from.getMySecretKey());
          KDCSendEncryptedFrameFromUser fromUser = new
KDCSendEncryptedFrameFromUser(encryptedKey,
                  encryptedNonce, encryptedLifetime, encryptedIdTo);
```

Краирање на KDCEncryptedFrameToUser со што се енкриптираат потребните информации за корисникот кој треба да ја прими порака и проследување на сесискиок клуч енкриптиран со неговиот клуч.

```
 byte[] encryptedKeyTo = AES.encrypt(sessionKey.getEncoded(), to.getMySecretKey());
          byte[] encryptedIdFrom = AES.encrypt(from.getId().getBytes(), to.getMySecretKey());
          byte[] encryptedLifetimeTo =
AES.encrypt(Utils.longToByteArray(KDCRepository.miliseconds), to.getMySecretKey());
          KDCSendEncryptedFrameToUser toUser = new
KDCSendEncryptedFrameToUser(encryptedKeyTo, encryptedIdFrom, encryptedLifetimeTo);
          KDCSendEncryptedFrame frame = new KDCSendEncryptedFrame(fromUser, toUser);
```

Наредна итерација се проследува енкапсулирана рамка до сервисот за корисници кои ја проследува рамката до соодветните корисници за верификација.
Притоа се креира и timestamp потребен за верификација од страна на корисникот кој ја добива пораката.

```
@Override
public boolean receiveFromKDC(String userId, KDCSendEncryptedFrame frame) throws
UserNotFoundException, IOException {
    User user = findById(userId);
    boolean isVerified = user.verifyFrameFromKDC(frame);
    String toId = Utils.fromEncryptedBytesToString(frame.getFrameFromUser().getToId(),
user.getMySecretKey());
    User userTo = findById(toId);
    ZonedDateTime timestamp = ZonedDateTime.now(ZoneId.systemDefault());
    Long epochSeconds = timestamp.toEpochSecond();
    byte[] ts = Utils.longToByteArray(epochSeconds);
    byte[] encTS = AES.encrypt(ts, userTo.getMySecretKey());
    boolean isVerifiedTo = userTo.verifyFrameToKDC(frame.getToUser(), encTS);
    return isVerified && isVerifiedTo;
}
```

Методи за верификација на иницијатор на праќање на порака.
Методите се наоѓаат во User класата.

```
public boolean verifyFrameFromKDC(KDCSendEncryptedFrame frame){
    return verifyNonce(frame.getFrameFromUser().getNonce())&&
            verifyLifetime(frame.getFrameFromUser().getLifetime());

}

private boolean verifyNonce(byte[] nonceEncrypted){
    byte [] nonceBytes  = AES.decrypt(nonceEncrypted,mySecretKey);
    return new String(nonceBytes).equals(this.nonce);
}



private boolean verifyLifetime(byte[] lifetimeEncrypted){
    byte[] lifetimeBytes = AES.decrypt(lifetimeEncrypted,mySecretKey);
    int lifetime = lifetimeBytes[0];
    return lifetime<=100;
}
```

Договор: lifetime не смее да биде поголем од 100 мс

```
public static long generateRandomLifetime(){
    return new SecureRandom().nextInt(100);
}
```