정보보안 과제1

20130940 장용훈

1. 암호화 복호화 구현

```
    // 암호화 복호화를 위한 클래스 선언
public class crypto2 {
    // 키생성을 위한 security.Key 생성
    private Key key ;
    //암호화 인지 복호화인지 알기위해 check 선언 , iv를 만들기 위해서이다.
    private int check;
    // 인스턴스 생성시 키를 초기화
    public crypto2(Key key) {
        // TODO Auto-generated constructor stub
        this.key = key;
    }
    // 암호화 메서드 crypt 활용
    public void encrypt(File source, File dest) throws Exception {
        // iv를 생성하기 위한 check
        check =1;
        crypt(Cipher.ENCRYPT_MODE, source, dest);
    }
    // 복호화 메서드 crypt 활용
    public void decrypt(File source, File dest) throws Exception {
        // iv를 생성하기 위한 check
        check =2;
        crypt(Cipher.DECRYPT_MODE, source, dest);
    }

// 실질적인 암호화와 복호화가 이루어지는 메서드(MODE만 다르고 적용되는 원리는 동일하기에 하나의 메서드 활용)
private void crypt(int mode, File source, File dest) throws Exception {
    // cipher 인스턴스 생성 AES 알고리즘, CBC, PKCS5 패딩
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    IvParameterSpec iv;
    // 동일한 iv를 생성하기 위해 check 변수 활용
    if(check ==1)  {
        // 출력파일을 iv로 설정
        String dest2 = dest.toString();
        // 출력파일의 길이가 16보다 작다면 , 길이를 16으로 맞춰줌. iv는 암호화 되지 않아도 되기에 가능함.
        while(dest2.length()<16) {
            dest2 +=0;
        }
        // dest2로 IvParameterSpec을 만들어줌
        iv = new IvParameterSpec(dest2.toString().getBytes("UTF-8"));
    }
```

```
else {
    //check가 2인경우 동일한 iv를 위해 source(암호화 할때 dest 파일)과 동일하게 만들어 준다.
    String dest2 = source.toString();
    // 출력파일의 길이가 16보다 작다면 , 길이를 16으로 맞춰줌. iv는 암호화 되지 않아도 되기에 가능함.
    while(dest2.length()<16) {
        dest2 +=0;
    }
    // dest2로 IvParameterSpec을 만들어줌
    iv = new IvParameterSpec(dest2.toString().getBytes("UTF-8"));
}
// cipher를 mode 값과 key 값 iv 값으로 만들어줌.
cipher.init(mode, key,iv);
InputStream input = null;
OutputStream output = null;

 try {
     // 파일을 읽고 쓰기 위해 선언
     input = new BufferedInputStream(new FileInputStream(source));
     output = new BufferedOutputStream(new FileOutputStream(dest));
     // 한번에 1024바이트씩 읽는다.
     byte[] buffer = new byte[1024];
     int read = -1;
     // 파일을 읽고, encryption 혹은 decryption하여 dest 에 써줌
     while ((read = input.read(buffer)) != -1) {
         output.write(cipher.update(buffer, 0, read));
     }
     // 버퍼에 있는 나머지 부분을 dest에 써줌
     output.write(cipher.doFinal());
```

```
            output.write(cipher.doFinal());
            // 파일을 닫아준다.
        } finally {
            if (output != null) {
                try {
                    output.close();
                } catch (IOException ie) {
                }
            }
            if (input != null) {
                try {
                    input.close();
                } catch (IOException ie) {
                }
            }
        }
    }
}
```

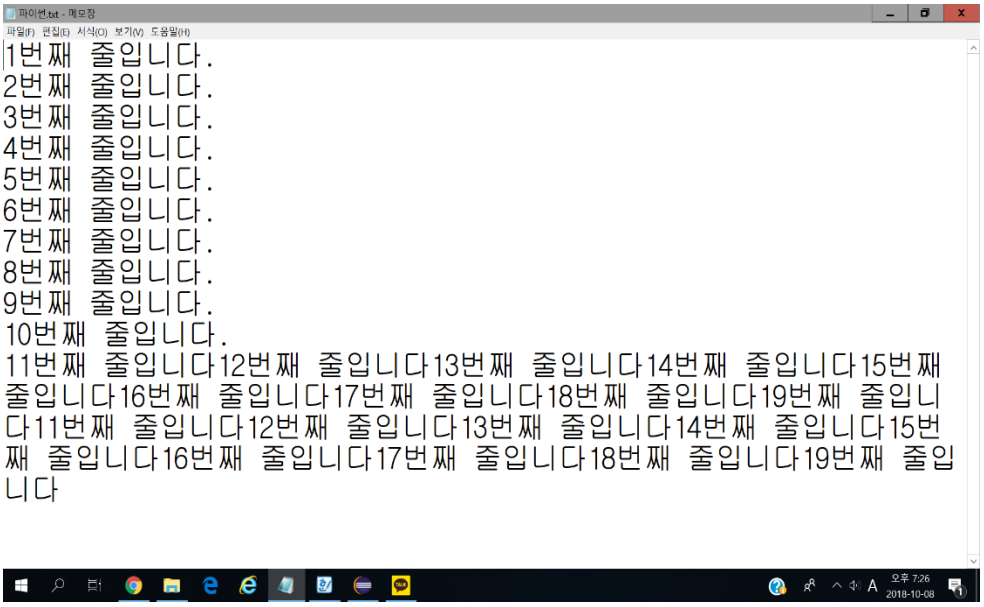
1-2) 메인 클래스-실행부분

```
public class crypto {

    public static void main(String[] args) throws Exception {
        // TODO Auto-generated method stub
        Scanner scan = new Scanner(System.in);
        String sourceFile;
        String dSourceFile;
        String destFile ;
        String destFile2 ;
        String password;
        int check;
        // 선택을 위해 주는 제어문이다.
        System.out.println("1번은 encryt 2번은 decrypt 3번은 종료:");
        check = scan.nextInt();
        // 암호화가 이루어진다.
        if(check == 1) {
        System.out.println("파일 입력: ");
        sourceFile = scan.next();
        // 암호의 입력 값이 16자리, 24자리, 32자리 만 가능하다. 만약 이를 어긴다면 exception이 발생한다.
        System.out.println("암호 입력: (16/24/32)");
        password = scan.next();
        System.out.println("암호화 파일이름 입력: ");
        destFile = scan.next();
```

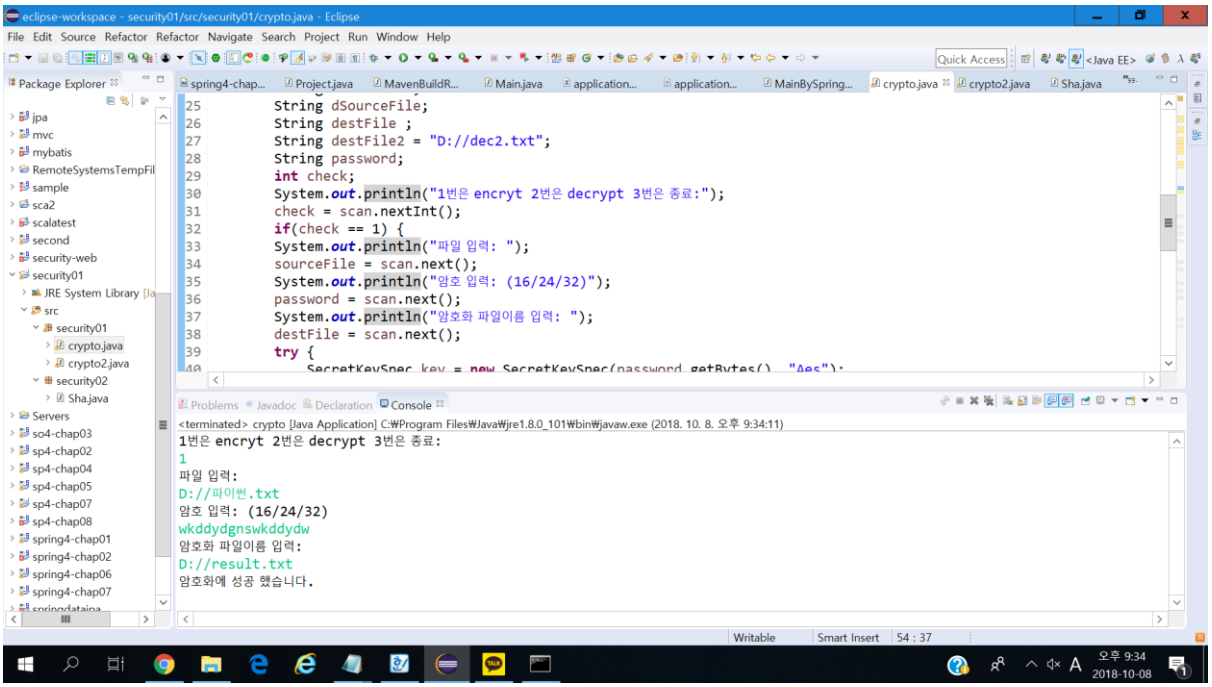
```
try {
    // 입력 받은 암호값을 이용하여 SecretKeySpec을 만든다. 이때 Aes알고리즘을 이용하여 생성한다.
    SecretKeySpec key = new SecretKeySpec(password.getBytes(), "Aes");
    // 키 값을 이용하여 인스턴스를 생성한다.
    crypto2 cry = new crypto2(key);
    // 암호화 진행
    cry.encrypt(new File(sourceFile), new File(destFile));
    // 암호화가 성공한다면 exception이 발생하지 않는다. 암호화 후 종료 된다.
    System.out.println("암호화에 성공 했습니다.");
} catch(Exception e) {
    // 암호화가 실패해서 exception이 발생한다. 메시지 출력 후 종료한다.
    System.out.println(e+" 암호화에 실패 했습니다.");
}


        }else if(check == 2) {
            System.out.println("복호화 할 파일명을 입력하세요: ");
            dSourceFile = scan.next();
            // 암호의 입력 값이 16자리, 24자리, 32자리 만 가능하다. 만약 이를 어긴다면 exception이 발생한다.
            System.out.println("암호를 입력하세요: (16/24/32)");
            password = scan.next();
            System.out.println("저장 될 경로를 입력하세요: ");
            destFile2 = scan.next();
            // 입력 받은 암호값을 이용하여 SecretKeySpec을 만든다. 이때 Aes알고리즘을 이용하여 생성한다.
            SecretKeySpec key = new SecretKeySpec(password.getBytes(), "Aes");
            // 키 값을 이용하여 인스턴스를 생성한다.
            crypto2 dcry = new crypto2(key);
            try {
            // 복호화를 수행한다.
            dcry.decrypt(new File(dSourceFile), new File(destFile2));
            // 복호화가 성공한다면 exception이 발생하지 않는다. 복호화 후 종료 된다.
            System.out.println("복호화에 성공 했습니다.");
            } catch(Exception e) {
            // 복호화가 실패해서 exception이 발생한다. 메시지 출력 후 종료한다.
                System.out.println(e+" 잘못된 암호 입니다.");
            }
        }
    }

}
```

1-3) 결과 부분

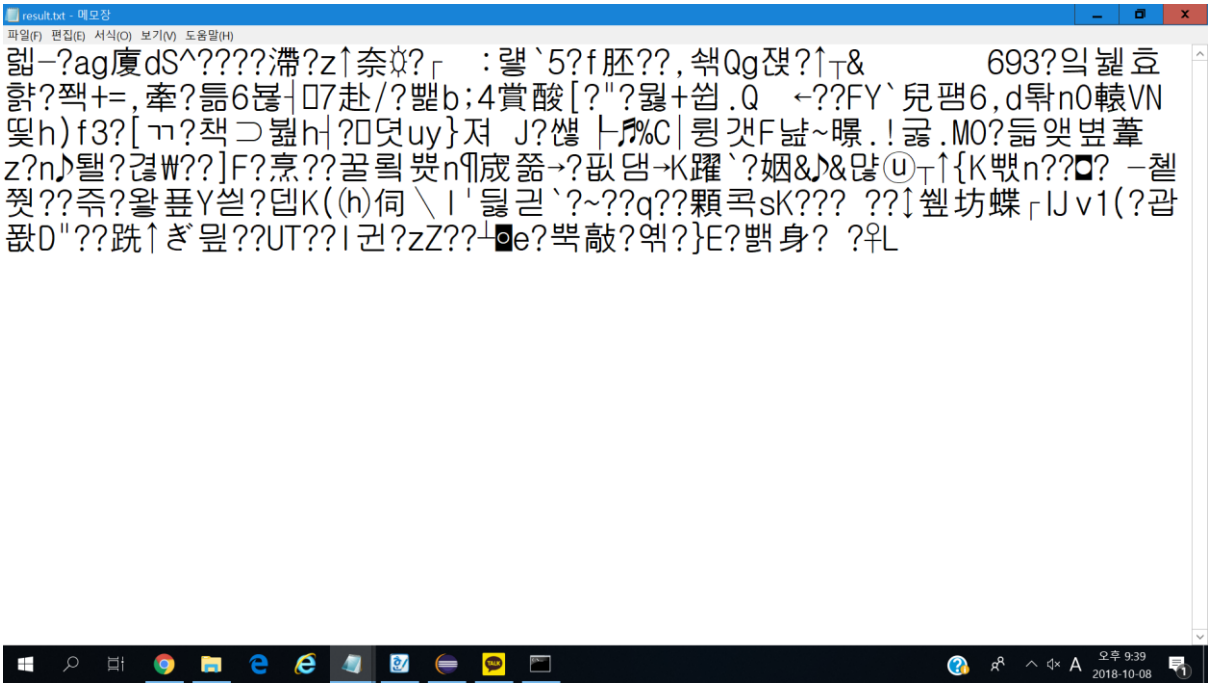
암호화 전 파일



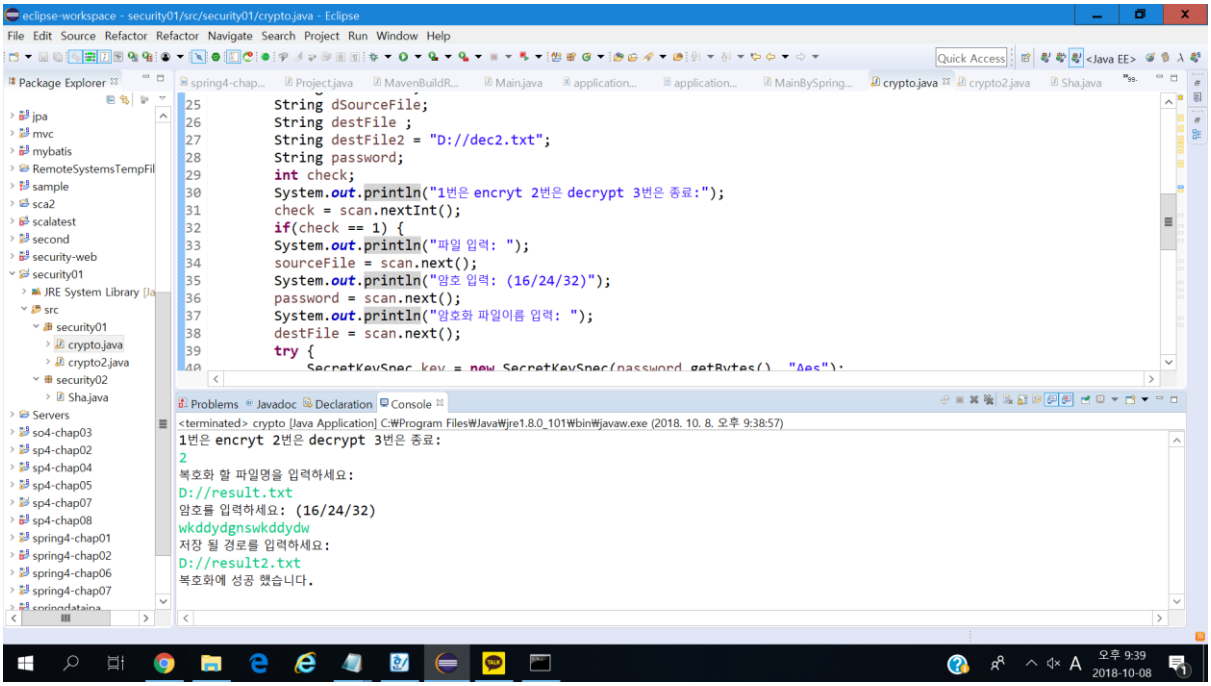
암호화 과정



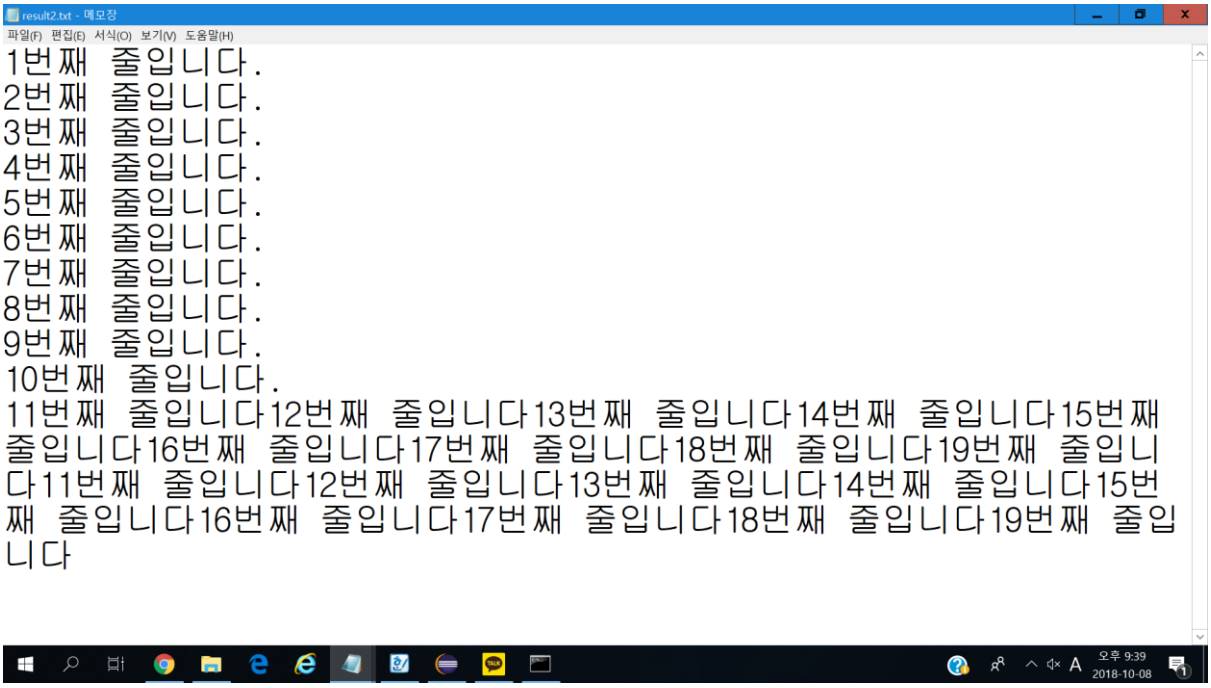
암호화에 성공한 파일



복호화 과정



복호화된 파일 확인

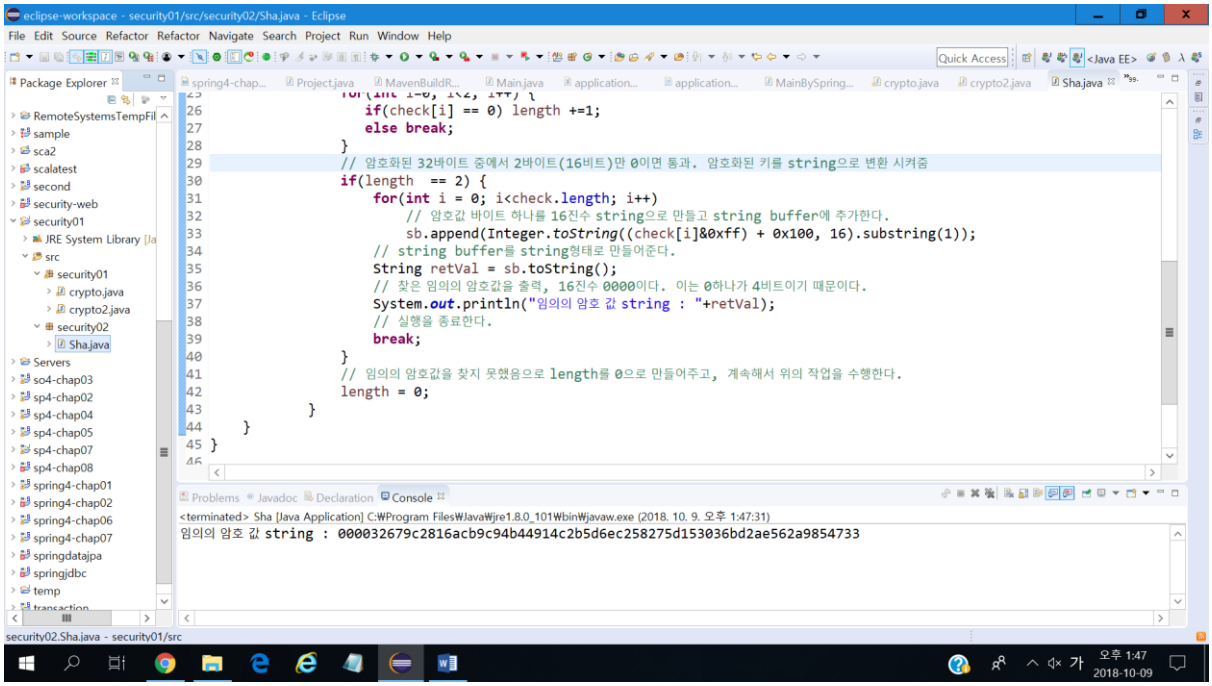


2. 임의의 SHA-256 을 생성해서, 연속된 0이 16비트 동안 나오는 임의의 암호 찾기

```
public class Sha {

    public static void main(String[] args) throws NoSuchAlgorithmException {
            int length  = 0;
            while(true) {
                // SHA-256을 이용하는 MessageDigest 인스턴스 생성
                MessageDigest ms = MessageDigest.getInstance("SHA-256");
                // 임의의 안전한 난수를 생성을 위해 SecureRandom 클래스 선언
                SecureRandom random = new SecureRandom();
                byte[] salt = new byte[256];
                // 256바이트의 랜덤 키 값 생성
                random.nextBytes(salt);
                // 랜덤 키 값으로 비밀키 생성
                ms.update(salt);
                // 암호화 된 키값을 얻음
                byte[] check = ms.digest();
                StringBuffer sb = new StringBuffer();
               // 키에서 0이 나올때마다 length를 계속 더해준다.
                for(int i=0; i<2; i++) {
                   if(check[i] == 0) length +=1;
                   else break;
                }
```

```
                // 암호화된 32바이트 중에서 2바이트(16비트)만 0이면 통과. 암호화된 키를 string으로 변환 시켜줌
                if(length  == 2) {
                    for(int i = 0; i<check.length; i++)
                        // 암호값 바이트 하나를 16진수 string으로 만들고 string buffer에 추가한다.
                        sb.append(Integer.toString((check[i]&0xff) + 0x100, 16).substring(1));
                    // string buffer를 string형태로 만들어준다.
                    String retVal = sb.toString();
                    // 찾은 임의의 암호값을 출력, 16진수 0000이다. 이는 0하나가 4비트이기 때문이다.
                    System.out.println("임의의 암호 값 string : "+retVal);
                    // 실행을 종료한다.
                    break;
                }
                // 임의의 암호값을 찾지 못했음으로 length를 0으로 만들어주고, 계속해서 위의 작업을 수행한다.
                length = 0;
            }
        }
    }
```

결과값 출력



임의의 암호 값 **string : 000032679c2816acb9c94b44914c2b5d6ec258275d153036bd2ae562a9854733**

3. 소견

수업시간에 배운 알고리즘 SHA-256, PKCS5 패딩 IV를 포함한 CBC 이를 AES에 적용하여 만들었다. 생각보다 어렵지 않았고 재미있었다. 참고부분과 거의 유사하게 만들었지만 일부 다른 점이 있었다. 첫번째로 암호화와 복호와의 과정에서 동일한 메서드를 활용 했다. MODE만 다를 뿐, 적용되는 과정은 동일하기 때문이다. 따라서 다음과 같이 선언하여 활용 하였다.

```
// 암호화 메서드 crypt 활용
public void encrypt(File source, File dest) throws Exception {
    // iv를 생성하기 위한 check
    check =1;
    crypt(Cipher.ENCRYPT_MODE, source, dest);
}
// 복호화 메서드 crypt 활용
public void decrypt(File source, File dest) throws Exception {
    // iv를 생성하기 위한 check
    check =2;
    crypt(Cipher.DECRYPT_MODE, source, dest);
}
```

crypt의 선언부는 다음과 같다.

```
private void crypt(int mode, File source, File dest) throws Exception
```

cipher를 이용하여 초기화 할 때, mode라는 파라메터에 따라 실질적으로 암호화할지 복호화 할지가 결정이 된다. 그리고 init 메서드에 Initialization vector를 추가할 수 있는 파라메터가 더해준다..

```
cipher.init(mode, key,iv);
```

그렇기에 cipher에 initialization vector를 추가하는 것이 수월 해졌다. 그런데 iv를 만들 때, 어떻게 초기화 할지 고민하다가, 암호화와 복호화에 쓰는 공통된 파일이름을 이용하여 iv를 만들었다. 그리고 iv의 사이즈가 정해져 있어서(16,32) 등 이를 충족하지 못하면 익셉션이 발생을 하게 되었다. 따라서 다음 사진과 같이 만들었다.

```
String dest2 = dest.toString();
// 출력파일의 길이가 16보다 작다면 , 길이를 16으로 맞춰줌. iv는 암호화 되지 않아도 되기에 가능함.
while(dest2.length()<16) {
    dest2 +=0;
}
// dest2로 IvParameterSpec을 만들어줌
iv = new IvParameterSpec(dest2.toString().getBytes("UTF-8"));
```

따라서 임의로 목표한 길이에 도달할 때 까지 0을 추가해주어서 이러한 문제를 해결했다. 다만 비밀번호의 경우에는 보안을 위해 위와 같이 추가해주지 않고 16자리에 맡게 생성하였다. 왜냐하면 iv의 경우에는 실제로 암호화하지 않기 때문이다. 그리고 나머지 부분은 InputStream, OutputStream을 이용하여 파일의 입출력을 수행했다는 것이 특이점이다. 두번째 과제는 SHA-256을 이용하여 임의의 값을 만들고 조건을 만족하는 값을 찾아냈다. 조건이 처음 16비트가 0을 만족하는 수(2비트가 조건에 맞는 것을 찾는 것)를 찾는 것이기에 다음과 같이 만들었다.

```
for(int i=0; i<2; i++) {
    if(check[i] == 0) length +=1;
    else break;
}
```

2번에서는 messageDigest로 암호를 만들면 32바이트의 암호가 나오는데 이는 32*8 = 256비트이기에 맞게 나온 것이다. 이 후 조건을 만족했을 때는 출력을 위해 이 바이트를 16진수로 만들어서 출력을 해주었다. 다음과 같다.

```
sb.append(Integer.toString((check[i]&0xff) + 0x100, 16).substring(1));
```

바이트를 16진수 수 255와 and 연산하여 0x100을 더한다. 이는 9자리의 수를 1로 만들어 주기 위해서이다. 그후 substring(1)은 상위 8비트를 무시하게 위해서 나타낸 것이다.

그 결과 아래와 같은 임의의 암호 값을 찾았다.

임의의 암호 값 `string : 000032679c2816acb9c94b44914c2b5d6ec258275d153036bd2ae562a9854733`