305 – Mission Impossible

Team Information

Team Name: ISEGYE_IDOL

Team Member: Eungchang Lee, Sojeong Kim, Mingyu Seong, Donghyun HA

Email Address: dfc-isegyeidol@googlegroups.com

Instructions

Description The police arrested a terrorist and seized his smartphone. They acquired an APK, and data that is expected to receive a secret mission. On the application, partial data is shown in plain text but the rest of them are found to be encrypted text. Be prepared for the next terror by decoding the message completely.

Target	Hash (MD5)
dfcm.zip	b26d85fc1edd8c6e99bf32d9205305b7

Questions

1. Identify the server address to which the APP requested the public key to exchange the key. (50 points)
2. Identify the password used to encrypt the database file (encrypted.db). (50 points)
3. Identify an algorithm used to exchange the key and describe why you think it is the algorithm used to exchange the key. (100 points)
4. Identify a key used to encrypt/decrypt messages from the given databases, and the hidden mission using the key. (100 points)

Teams must:

- Develop and document the step-by-step approach used to solve this

problem to allow another examiner to replicate team actions and results.

- Specify all tools used in deriving the conclusion(s).

Tools used:

Name:	JADX	Publisher:	skylot
Version:	1.4.3		
URL:	https://github.com/skylot/jadx		

Name:	Frida	Publisher:	Frida
Version:	15.2.2		
URL:	https://frida.re/		

Name:	SQLiteDatabaseBrowser	Publisher:	sqlitebrowser
Version:	3.12.2		
URL:	https://sqlitebrowser.org/		

Step-by-step methodology:

1. Identify the server address to which the APP requested the public key to exchange the key.

안드로이드 APK 분석 프로그램인 Jadx를 이용하여 주어진 APK 파일을 디컴파일하였다. [org.df.string] 클래스 내에는 네이티브 라이브러리 "string"에 작성 되어있는 getURL 함수가 존재한다.

```
package org.df.string;

/* loaded from: classes.dex */
public class string {
    static {
        System.loadLibrary("string");
    }

    public static native String getURL();
}
```

[그림 1] org.df.string 클래스

해당 함수의 참조를 따라가 보면 z1.e.d()에서 사용된다. string.getURL() 함수가 java.net.URL의 인자로 전달되고 있다. Java.net.URL은 string 형태의 url을 받아 URL 객체를 만들어 openConnection과 같은 함수를 통해 원격 통신을 할 수 있도록 한다.

```
    public JSONObject d() {
        try {
            HttpURLConnection httpURLConnection = (HttpURLConnection) new URL(string.getURL()).openConnection();
            httpURLConnection.setRequestMethod("GET");
            if (httpURLConnection.getResponseCode() != 200) {
                return null;
            }
            BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(httpURLConnection.
getInputStream(), "UTF-8"));
            StringBuffer stringBuffer = new StringBuffer();
            while (true) {
                String readLine = bufferedReader.readLine();
```

[그림 2] z1.e.d() 함수

따라서 string.getURL을 통해 특정 서버의 url을 받아와서 해당 서버에 GET요청을 보낸 후 데이터를 받아오는 방식으로 통신을 하게 된다.

getURL 함수의 내용을 확인하기 위해서 동적분석 방식을 이용하였다. 아래의 Frida 스크립트를 통해 해당 함수를 후킹하여 반환값을 확인하였다.

```
import frida, sys

jscode = """
Java.perform(function(){
    console.log(
        Java.use('org.df.string.string')['getURL']()
        );
});
"""
device = frida.get_usb_device(1)
process = device.spawn(['org.df.challenge'])
session = device.attach(process)
script = session.create_script(jscode)
script.load()
device.resume(process)
sys.stdin.read()
```

[표 1] getURL 후킹 스크립트

후킹 스크립트의 반환값은 다음 url과 같고, 접속 결과 [key.json] 파일을 확인하였다.

- http://61.97.186.252:8080/key.json


```
{
    "pub_key":  [160, 162, 143, 209, 85, 167, 200, 9, 34, 45, 218, 128, 240, 69, 170, 84, 60, 58, 34, 153, 117, 27, 149, 154, 92, 186, 85, 214, 146, 201,
221, 106, 0, 184, 96, 75, 195, 144, 195, 142, 54, 51, 168, 60, 71, 173, 117, 34, 68, 244, 89, 76, 52, 56, 93, 228, 93, 75, 229, 107, 17, 100, 107, 60, 181,
79, 196, 53, 149, 27, 201, 127, 16, 171, 153, 179, 180, 104, 172, 128, 162, 62, 246, 79, 37, 220, 125, 126, 226, 83, 188, 42, 70, 230, 150, 194, 155, 75,
191, 17, 134, 127, 47, 240, 193, 53, 115, 135, 97, 75, 99, 138, 156, 52, 202, 133, 57, 231, 184, 71, 17, 194, 9, 181, 119, 2, 41, 196, 4, 41, 74, 33, 119,
65, 0, 166, 186, 41, 178, 0, 20, 70, 216, 67, 64, 48, 55, 68, 73, 197, 92, 176, 4, 46, 28, 194, 124, 91, 129, 17, 229, 196, 158, 113, 70, 230, 136, 126, 109,
217, 160, 207, 37, 123, 199, 3, 208, 244, 44, 148, 18, 149, 172, 213, 19, 89, 119, 65, 143, 184, 177, 73, 124, 200, 46, 151, 89, 200, 249, 235, 87, 242, 225,
138, 45, 84, 49, 64, 152, 40, 185, 73, 199, 127, 152, 16, 225, 74, 190, 79, 197, 125, 124, 99, 133, 91, 155, 183, 152, 219, 112, 25, 107, 29, 146, 195, 181,
182, 132, 35, 42, 247, 109, 88, 203, 130, 6, 220, 81, 2, 5, 24, 223, 40, 87, 160, 33, 22, 110, 48, 146, 66, 217, 62, 216, 227, 33, 242, 203, 183, 227, 188,
171, 160, 138, 171, 28, 215, 35, 24, 87, 172, 197, 240, 68, 85, 214, 141, 252, 243, 89, 64, 40, 43, 115, 213, 67, 90, 165, 158, 107, 138, 29, 212, 251, 104,
211, 27, 10, 162, 187, 207, 195, 48, 52, 240, 120, 181, 241, 121, 90, 223, 136, 83, 36, 99, 47, 138, 44, 100, 46, 96, 114, 97, 230, 156, 220, 56, 68, 30, 42,
145, 148, 137, 204, 81, 195, 125, 206, 19, 188, 112, 211, 37, 230, 83, 165, 255, 35, 123, 70, 133, 141, 154, 118, 75, 101, 216, 123, 166, 19, 197, 240, 245,
65, 243, 246, 98, 91, 57, 162, 108, 193, 138, 85, 211, 78, 39, 171, 156, 49, 130, 173, 49, 217, 63, 145, 56, 93, 171, 107, 110, 228, 8, 136, 51, 10, 105,
229, 68, 166, 103, 53, 76, 184, 19, 126, 23, 132, 73, 107, 117, 178, 51, 224, 13, 218, 101, 34, 20, 225, 116, 115, 132, 172, 155, 22, 34, 101, 12, 22, 244,
188, 171, 185, 135, 116, 45, 123, 20, 2, 6, 131, 125, 220, 35, 125, 26, 1, 53, 144, 27, 246, 33, 46, 131, 113, 95, 56, 165, 112, 32, 163, 159, 146, 9, 49,
231, 129, 55, 108, 226, 69, 91, 73, 77, 172, 82, 11, 72, 115, 97, 43, 1, 205, 42, 170, 73, 136, 105, 86, 87, 37, 26, 198, 226, 21, 27, 198, 103, 38, 10, 97,
183, 226, 182, 208, 66, 158, 231, 68, 113, 213, 227, 97, 176, 16, 98, 221, 210, 18, 70, 225, 132, 180, 7, 97, 209, 24, 19, 225, 98, 51, 81, 244, 4, 195, 172,
43, 255, 216, 9, 11, 187, 187, 143, 164, 159, 123, 117, 62, 59, 243, 100, 196, 145, 164, 82, 171, 184, 175, 48, 195, 23, 247, 25, 70, 54, 204, 214, 108, 10,
242, 107, 121, 166, 9, 42, 165, 153, 145, 150, 242, 195, 117, 83, 167, 161, 208, 37, 214, 23, 19, 169, 64, 165, 109, 91, 73, 49, 139, 194, 24, 188, 138, 201,
156, 94, 46, 19, 147, 217, 197, 177, 146, 16, 155, 160, 6, 29, 122, 25, 4, 237, 114, 56, 158, 67, 155, 111, 18, 70, 120, 57, 204, 130, 71, 112, 98, 214, 137,
2, 252, 11, 46, 52, 202, 153, 232, 103, 227, 178, 52, 48, 135, 71, 207, 211, 104, 198, 88, 96, 6, 84, 82, 213, 10, 198, 166, 233, 45, 107, 156, 186, 105, 54,
195, 141, 212, 116, 63, 115, 71, 102, 131, 1, 232, 199, 150, 235, 151, 8, 89, 7, 177, 128, 185, 0, 128, 38, 129, 14, 235, 60, 37, 144, 20, 94, 193, 182, 147,
212, 155, 59, 164, 126, 154, 70, 18, 187, 216, 137, 159, 217, 33, 110, 150, 39, 84, 27, 145, 172, 128, 3, 90, 162, 162, 179, 133, 118, 140, 202, 170, 14,
188, 34, 14, 154, 155, 148, 88, 149, 246, 153, 146, 20, 246, 145, 150, 151, 111, 85, 41, 180, 111, 160, 72, 204, 249, 147, 251, 89, 7, 22, 118, 194, 93, 187,
69, 92, 145, 56, 207, 185, 45, 79, 135, 110, 205, 242, 127, 132, 56, 33, 211, 16, 85, 214, 150, 176, 176, 151, 166, 19, 211, 93, 147, 73, 193, 198, 112, 126,
4, 43, 63, 99, 82, 41, 124, 248, 37, 88, 35, 184, 196, 139, 11, 143, 83, 114, 96, 3, 60, 107, 28, 196, 158, 130, 197, 74, 182, 106, 34, 16, 199, 144, 51, 86,
67, 114, 135, 204, 214, 54, 38, 81, 3, 238, 230, 121, 196, 251, 132, 60, 118, 18, 226, 219, 61, 177, 199, 65, 127, 234, 187, 36, 106, 179, 177, 68, 146, 149,
132, 141, 78, 20, 33, 43, 52, 145, 107, 235, 155, 136, 177, 33, 1, 28, 195, 3, 28, 78, 170, 183, 107, 40, 88, 11, 78, 57, 51, 158, 82, 72, 45, 128, 140, 125,
70, 94, 190, 242, 35, 236, 19, 40, 99, 151, 144, 78, 184, 183, 75, 11, 162, 14, 202, 196, 57, 166, 19, 45, 115, 97, 20, 186, 73, 91, 211, 9, 123, 117, 194,
157, 245, 132, 23, 43, 177, 86, 138, 205, 210, 69, 144, 79, 210, 75, 185, 19, 192, 119, 112, 118, 62, 113, 98, 48, 118, 63, 97, 200, 190, 236, 151, 205, 107,
7, 150, 240, 118, 197, 248, 153, 206, 3, 131, 31, 187, 161, 151, 208, 245, 1, 233, 235, 206, 48, 170, 13, 146, 123, 69, 144, 9, 188, 114, 101, 162, 90, 194,
31, 88, 106, 146, 80, 203, 116, 109, 216, 143, 216, 21, 91, 46, 27, 68, 11, 67, 55, 220, 115, 174, 134, 187, 52, 164, 186, 114, 21, 22, 194, 150, 163, 160,
22, 116, 5, 86, 214, 97, 106, 186, 147, 41, 133, 142, 251, 211, 131, 139, 234, 145, 74, 69, 162, 200, 76, 27, 15, 212, 173, 78, 50, 131, 132, 74, 204, 118,
178, 131, 13, 146, 1, 164, 248, 121, 235, 85, 117, 165, 199, 91, 65, 23, 86, 124, 84, 32, 10, 148, 136, 170, 152, 186, 130, 25, 166, 144, 180, 174, 57, 140,
121, 15, 208, 12, 196, 166, 126, 2, 124, 54, 219, 84, 131, 25, 107, 142, 210, 49, 132, 26, 229, 45, 216, 208, 85, 122, 106, 170, 74, 113, 106, 24, 68, 15,
114, 70, 43, 180, 206, 67, 248, 27, 172, 168, 151, 101, 1, 130, 207, 130, 58]
```


[그림 3] key.json

2. Identify the password used to encrypt the database file (encrypted.db).

해당 앱에서는 database 파일을 암호화해서 저장하는 방식을 사용하고 있다. DB 파일명은 [encrypted.db]로 Jadx 프로그램에서 검색하여 해당 DB를 로드하는 부분을 발견했다.

```
 9 public final class b extends SQLiteOpenHelper {
10
11     /* renamed from: a */
12     public SQLiteDatabase f3777a;
13
14     public b(Context context, String str) {
15         super(context, "encrypted.db", null, 1);
16         this.f3777a = null;
17         SQLiteDatabase.loadLibs(context);
18         try {
19             File file = new File("/data/data/org.df.challenge/databases");
20             if (!file.exists()) {
21                 file.mkdirs();
22             }
23             SQLiteDatabase openOrCreateDatabase = SQLiteDatabase.openOrCreateDatabase(
   "/data/data/org.df.challenge/databases/encrypted.db", str, (SQLiteDatabase.CursorFactory) null);
24             this.f3777a = openOrCreateDatabase;
25             openOrCreateDatabase.execSQL("CREATE TABLE IF NOT EXISTS secret_key(encrypt_key TEXT)");
26             this.f3777a.execSQL("CREATE TABLE IF NOT EXISTS messages(message TEXT, timestamp DATETIME)");
27         } catch (Exception e3) {
28             e3.printStackTrace();
29         }
30     }
31
```

[그림 3] z2.b 클래스

DB를 실제로 로드하는 openOrCreateDatabase 함수의 명세는 sqlcipher의 소스코드를 참고하여 알아냈다.

```
1280    /**
1281     * Equivalent to openDatabase(file.getPath(), password, factory, CREATE_IF_NECESSARY, databaseHook).
1282     */
1283    public static SQLiteDatabase openOrCreateDatabase(File file, String password, CursorFactory factory, SQLiteDatabaseHook databaseHook) {
1284      return openOrCreateDatabase(file, password, factory, databaseHook, null);
1285    }
```

[그림 4] github code[1]

[그림 3] z2.b 클래스의 23번째 줄 openOrCreateDatabase의 두번째 인자인 str은 DB 비밀번호로 사용되고 있다.

[1] https://github.com/sqlcipher/android-database-sqlcipher/blob/master/android-database-sqlcipher/src/main/java/net/sqlcipher/database/SQLiteDatabase.java

str은 함수 b의 두 번째 인자로 받아오고 있다. b가 생성되는 부분을 추적해보면 getSharedPreferences를 이용하여 "app_preferences"의 "dGhpc2lzc2VjcmV0a2V5"를 키 값으로 하는 값을 받아 base64 디코드를 하여 비밀번호로 사용하는 것을 확인하였다.

```
    @Override // androidx.fragment.app.p, androidx.activity.ComponentActivity, w.g, android.app.Activity
    public final void onCreate(Bundle bundle) {
        Cursor rawQuery;
        super.onCreate(bundle);
        setContentView(R.layout.activity_main);
        StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder().permitAll().build());
        String string = getSharedPreferences("app_preferences", 0).getString("dGhpc2lzc2VjcmV0a2V5",
"app_preferences");
        a aVar = new a();
        b bVar = new b(this, new String(Base64.decode(string.getBytes(StandardCharsets.UTF_8), 0)));
```

[그림 5] MainActivity.onCreate()

SharedPreferences는 APK 파일 내부에 저장되므로 해당 APK 파일을 압축 해제하여 [/data/shared_prefs/app_preferences.xml]에서 해당 값을 확인하였다.

```
1  <?xml version='1.0' encoding='utf-8' standalone='yes' ?>
2  <map>
3      <string name="dGhpc2lzc2VjcmV0a2V5">QCFwYXNzd29yZCFA</string>
4  </map>
5
```

[그림 6] /data/shared_prefs/app_preferences.xml

"QCFwYXNzd29yZCFA"를 base64 디코드하여 "**@!password!@**"라는 패스워드를 확인했다..

3. Identify an algorithm used to exchange the key and describe why you think it is the algorithm used to exchange the key.

MainActivity의 onCreate은 키 교환 과정이 포함되어 있다.

먼저 [그림 7] 34번 줄에서 DB 파일에 접근하여 secret_key라는 테이블의 정보를 받아온다. 해당 테이블에 어떠한 정보도 없으면 키교환을 시도하게 된다. 그 과정은 [그림 7] 39번 줄에서 #1에서 찾아낸 원격 서버로 요청을 보내 pub_key를 받아오면서 시작된다.

```
32          a aVar = new a();
33          b bVar = new b(this, new String(Base64.decode(string.getBytes(StandardCharsets.UTF_8), 0)));
34          Cursor rawQuery2 = bVar.f3777a.rawQuery("SELECT  * FROM secret_key", (String[]) null);
35          int count = rawQuery2.getCount();
36          rawQuery2.close();
37          if (!(count > 0)) {
38              try {
39                  JSONArray jSONArray = new e().d().getJSONArray("pub_key");
40                  for (int i3 = 0; i3 < jSONArray.length(); i3++) {
41                      aVar.f3775a[i3] = (byte) jSONArray.optInt(i3);
42                  }
43              } catch (Exception e3) {
44                  e3.printStackTrace();
45              }
46              crypto.encrypt(aVar.c, aVar.f3776b, aVar.f3775a);
47              String encodeToString = Base64.encodeToString(aVar.f3776b, 0);
48              try {
49                  bVar.f3777a.execSQL("INSERT INTO secret_key(encrypt_key) VALUES ('" + encodeToString + "')");
50              } catch (Exception e4) {
51                  e4.printStackTrace();
52              }
53          } else {
```

[그림 7] MainActivity.onCreate()

해당 pub_key 정보를 aVar.f3775a에 저장하게 된다. pub_key를 저장한 이후 crypto.encrypt함수에 aVar.c, aVar.f3776b, aVar.f3775a를 인자로 전달하여 실행한다. 이후 aVar.f3776b의 값을 base64인코드 하여 DB에 저장한다.

```
1  package z2;
2
3  /* loaded from: classes.dex */
4  public final class a {
5
6      /* renamed from: a  reason: collision with root package name */
7      public byte[] f3775a = new byte[1184];
8
9      /* renamed from: b  reason: collision with root package name */
10     public byte[] f3776b = new byte[32];
11     public byte[] c = new byte[1088];
12
13     static {
14         "0123456789012345".getBytes();
15     }
16 }
```

[그림 8] z2.a

이때 encrypt에서 사용되었던 aVar.c, aVar.f3776b, aVar.f3775a의 크기는 각각 1088, 32, 1184 바이트이다. 해당 바이트들에 관련한 키 교환 알고리즘을 검색한 결과 Kyber768 방식을 이용하였을 때의 스펙과 동일한 것을 확인하였다.

KYBER768	
Sizes (in Bytes)	
sk:	2400 (or 32)
pk:	1184
ct:	1088

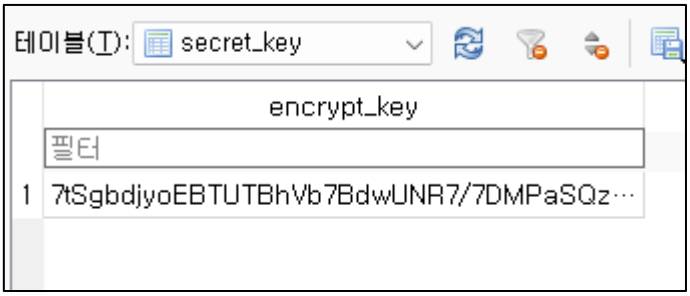
[그림 9] Kyber768 스펙[2]

따라서 해당 어플리케이션에서는 원격서버와 **Kyber768 키교환 알고리즘**을 사용하고 있음을 유추할 수 있다.

[2] https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf

4. Identify a key used to encrypt/decrypt messages from the given databases, and the hidden mission using the key.

해당 어플리케이션이 Kyber768 키교환을 사용하므로 앞서 보았던 aVar.c, aVar.f3776b, aVar.f3775a은 각각 ct, sk, pk임을 알 수 있다. 키 교환의 주체인 sk는 DB에 저장되어 있으므로 해당 값을 키로 복호화를 진행할 수 있다.



[그림 10] encrypted.db

해당하는 키 값은 "7tSgbdjyoEBTUTBhVb7BdwUNR7/7DMPaSQzm1Xjo7WM="으로 32바이트가 base64 인코딩 되어있다. 일반적인 32바이트 암호화인 AES256을 이용하여 messages 테이블의 내용을 복호화를 시도했지만 평문이 나오지 않아 특정 방식으로 키를 변형하였다고 유추하였다.

여러가지 방법을 시도해본 결과 aes128의 키로 sk의 앞 16바이트를 이용하고 나머지 16바이트는 iv로 사용하여 복호화를 진행하였고, 아래의 코드로 평문을 도출해냈다.

```
#-*-coding:utf-8-*-
import base64
from Crypto.Cipher import AES
iv = ""
BS = AES.block_size
class AESCipher:
    def __init__(self):
        self.key =
base64.b64decode('7tSgbdjyoEBTUTBhVb7BdwUNR7/7DMPaSQzm1Xjo7WM=')

    def decrypt(self, enc):
        enc = base64.b64decode(enc)
        iv = enc[:16]
        enc = enc[16:]
        print(len(iv))
        print(len(enc))
        cipher = AES.new(self.key, AES.MODE_CBC, iv)
```

```
        dec = cipher.decrypt(enc)
        return dec
    def decryptRaw(self, enc):
        cipher = AES.new(self.key, AES.MODE_ECB)
        dec = cipher.decrypt(enc)
        return dec

aes = AESCipher()
first =
aes.decrypt("NTqOT35NT3OTjKLha9iymLVwKAsrry55t8/F/kJaM3gVvAnUQYGTGvzSuy5ty+P+
")
second =
aes.decrypt('gv9ozC0drWIGLpCPh8ExcCcDoYCinNt2g7VaRaiz2arB43tRna/IvaKkNj1Pe8ZX
EorJl5jlkS2JdP7W19hzwQ==')
third =
aes.decrypt("ELRyqP82Lf+rzTsc/3vPhNlX1oq9nngKWu8rcyaxOJe9x/bBg3cFHfBJCPsA8xOv
cm6rnJFM47JLrK3I1ebLSuV1hqxtNfnXC8U+TQRt7Dc=")

print(first)
print(second)
print(third)
```

[표 2] 복호화 코드

최종적으로 복호화 코드를 실행하여 얻은 숨겨진 미션은 다음과 같다.

- **make a fuss in Seoul**
- **you have to destroy is 'N SEOUL TOWER**
- **l be executed on September 30, prove your loyalty**