

301 – What is the secret information

Team Information

Team Name : DogeCoin

Team Member : Dongbin Oh, Donghyun Kim, Donghyun Kim, Yeongwoong Kim

Email Address : dfc-dogecoin@naver.com

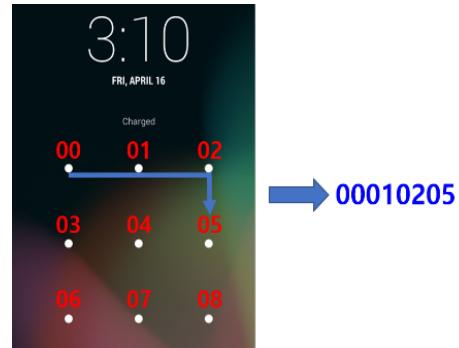
Instructions

Description You are an investigator. You had received intelligence that international drug dealer A was scheduled to enter Korea and meet with someone and secret information related to the meeting is stored on A's smartphone. The A was arrested at Incheon International Airport and A's smartphone was confiscated. After then, you dumped the NAND flash memory and collected /data partition image. As a result of interrogation of A, you identified that A purchased the smartphone in March 2014 and that secret information was encrypted with a security app. You also found out that the security app is using the screen lock number as the passcode.

Target	Hash (MD5)
data.img	d35c83baceed0ea602611d3939fc7eef

Questions

1. Identify all unlock methods that the A is using (PIN / Pattern / Password). (100 points)
2. Find all keys (numbers / pattern / string) to unlocking the screen. (If Pattern is used, write it down by referring to the coordinates in the figure below.) (100 points)



3. What is the A's secret information? (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:	SQLite DB browser	Publisher:	https://dbhub.io/
Version:	3.12.2		
URL:	https://sqlitebrowser.org/		

Name:	GestureCrack	Publisher:	KieronCraggs
Version:			
URL:	https://github.com/KieronCraggs/GestureCrack		

Name:	Hashcat	Publisher:	Hashcat
Version:	6.1.1		
URL:	https://hashcat.net/hashcat/		

Name:	Android Studio	Publisher:	Google developers
-------	----------------	------------	-------------------

Version:	202.7351085
URL:	https://developer.android.com/studio

Name:	Frida	Publisher:	Frida
Version:	14.2.18		
URL:	https://github.com/frida		

Name:	android-ks-decryptor	Publisher:	Synacktiv
Version:			
URL:	https://github.com/Synacktiv-contrib/stuffz/blob/master/android-ks-decryptor.py		

Name:	JADX	Publisher:	skylot
Version:	1.2.0		
URL:	https://github.com/skylot/jadx		

Name:	android-keystore-audit	Publisher:	FSecureLABS
Version:			
URL:	https://github.com/FSecureLABS/android-keystore-audit		

Step-by-step methodology:

1. Identify all unlock methods that the A is using (PIN / Pattern / Password). (100 points)

용의자는 Android의 화면 잠금을 위해 Pattern 뿐만 아니라 PIN과 Password를 모두 사용하였음. 이는 locksettings.db 파일과 비활당영역에서 추출한 locksettings.db 관련 정보 (locksettings.password_type)을 통해 확인 가능

문제에서 용의자는 2014년 3월에 구매한 안드로이드 폰을 사용한다고 언급하였다. 이를 통해 용의자가 사용하고 있는 핸드폰은 아래의 그림과 같이 최소 Kitkat 운영체제 이하를 사용한다는 것을 파악할 수 있다.

June 2014	Android 4.4W	20	Kitkat Watch
October 2013	Android 4.4	19	Kitkat
July 2013	Android 4.3	18	Jelly Bean
November 2012	Android 4.2-4.2.2	17	Jelly Bean
June 2012	Android 4.1-4.1.1	16	Jelly Bean
December 2011	Android 4.0.3-4.0.4	15	Ice Cream Sandwich
October 2011	Android 4.0-4.0.2	14	Ice Cream Sandwich
June 2011	Android 3.2	13	Honeycomb
May 2011	Android 3.1.x	12	Honeycomb

【그림 1】 안드로이드 버전 표

해당 버전 이하의 안드로이드의 경우 PIN / Pattern / Password 형식의 화면 잠금 방식을 모두 지원하고 “locksettings.db”, “password.key”, “Gesture.key”라는 파일을 통해 관련 내용을 확인할 수 있다. 해당 파일은 모두 /data/system/에 위치하여있다.

locksettings.db는 잠금 방식의 메타데이터를 저장하는 파일이다. 권한을 가진 사용자가 이 파일을 변조할 경우 기존 잠금 방식을 무시하고 원하는 잠금 방식을 사용하도록 지정할 수 있다.

locksettings.db는 Sqlite 형식을 가지고 있으며 아래의 그림과 같은 내용을 포함하고 있다.

테이블: locksettings

	_id	name	user	value
	필터	필터	필터	필터
1	1	lockscreen.disabled	0	1
2	2	migrated	0	true
3	3	lock_screen_owner_info_enabled	0	0
4	4	migrated_user_specific	0	true
5	5	lockscreen.password_salt	0	-9189886932...
6	14	lock_pattern_visible_pattern	0	1
7	32	lockscreen.password_type_alternate	0	0
8	34	lockscreen.passwordhistory	0	3198F1EA9F76...
9	35	lockscreen.patterneverchosen	0	1
10	36	lockscreen.password_type	0	65536
11	37	lock_pattern_autolock	0	1

[그림 2] locksettings.db 내부

Name column을 통해 locksetting.db에 포함된 잠금 방식의 정보들을 알 수 있다. 정보 중 본 문제에서 요구하는 모든 잠금 방식을 파악하기 위해서는 lockscreen.password_type을 분석해야한다. lockscreen.password_type은 DevicePolicyManager를 통하여 정의되어 있으며 세부 사항은 다음 URL¹에서 확인 가능하다.

또한 이전에 사용했던 잠금 방식이 비활당영역에 존재할 가능성을 고려하여, 비활당영역을 대상으로 아래와 같이 String Match 기반의 Carving을 시도하였다.

```
lockscreen.password_type327680(
lockscreen.password_type_alternate00
3lockscreen.password_salt8671682885601775202"
lock_pattern_visible_pattern1
lock_pattern_autolock0
lockscreen.passwordhistory"
lockscreen.patterneverchosen1
migrated_user_specifictrue
migratedtrue
lockscreen.disabled1
locksettings
58D61E291D102DFF5A5FF090F6F4628BC3C7BC8AF2BADF6669159E81476A8147F76435FB
SQLite format 3
SQLITE_FORMAT_3
```

[그림 3] String Match 기반 카빙 결과

SQLite Database의 Header Signature의 “SQLite format 3”이라는 글자 이외에도, locksettings.db에서 확인했던 값들을 확인할 수 있었다.

Strings data.img | grep lockscreen.password_type | sort | uniq 명령을 통해 Password_type로 사용되는 value들을 모두 파악한 결과는 아래와 같다. 사용하고 있는 Android 운영체제가 kitkat이므로 API 버전이 190이므로 DevicePolicyManger에 맞추어 내용을 정리할 수 있다.

¹ <https://developer.android.com/reference/android/app/admin/DevicePolicyManager>

- 65536 → PASSWORD_QUALITY_SOMETHING → Pattern
- 131072 → PASSWORD_QUALITY_NUMERIC → PIN
- 327680 → PASSWORD_QUALITY_ALPHANUMERIC → Password
- 0 → PASSWORD_QUALITY_UNSPECIFIED → None

즉, 피의자로부터 입수한 안드로이드 핸드폰은 현재 Pattern을 사용하고 있었으며, 과거에 PIN과 Password를 모두 사용한 흔적이 있다.

2. Find all keys (numbers / pattern / string) to unlocking the screen. (If Pattern is used, write it down by referring to the coordinates in the figure below.) (100 points)

PIN - 1111,2222

String - a111, ace111, ace222, ninano1

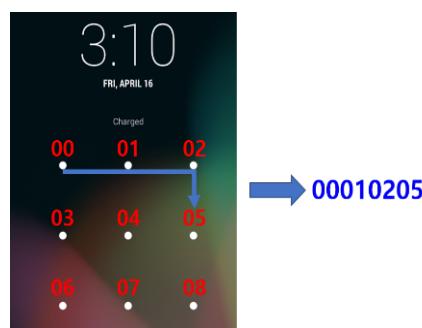
Pattern - 000102050807063004, 070401020508,

00010204060708, 000102050807063004

1번 문항을 통하여 피의자로부터 압수한 안드로이드 폰에서 PIN / Pattern / Password를 모두 사용하고 있는 것을 확인할 수 있었다. 2번 문항에서는 확인된 Password type을 기반으로 Password / PIN / Pattern를 직접 복구하는 것을 목표로 하고 있다.

- Pattern 복구

Pattern을 복구하기 위해서는 /data/system 의 “Gesture.key” 파일을 통해 가능하다. “Gesture.key”의 경우 패턴의 각 요소 값을 아래와 같이 hex 값으로 표현 후 이를 SHA1 hash 처리한 값이다. 아래의 그림과 같이 Pattern 이 00010205일 경우, “Gesture.key” 는 SHA1(0x00010205)에 해당하는 값을 가지고 있다.



[그림 4] 패턴 값 저장 형태

사용 가능한 패턴의 경우는 986,328가지로 한정적이므로 이에 대한 RainbowTable과 Crack script를 구할 수 있다. 아래는 gesturecrack이라는 도구를 사용하여 /data/system의 ‘Gesture.key’ 파일 내용을 추출 시도 하였

다.

```
(base) sin90@odongbin-ui-MacBookPro GestureCrack % python2 gesturecrack.py -r "4  
47fd4362cc0b65c1a2aae44316c3e2a33ba4fa9"  
  
The Lock Pattern code is [7, 4, 1, 2, 5, 8]  
For reference here is the grid (starting at 0 in the top left corner):  
  
| 0 | 1 | 2 |  
| 3 | 4 | 5 |  
| 6 | 7 | 8 |  
  
(base) sin90@odongbin-ui-MacBookPro GestureCrack %
```

[그림 5] Gesture.key 크랙

Gesturecrack을 통해 070401020508 이 현재 사용하고 있는 Pattern인 것을 확인할 수 있다. 추가적으로 1번 문항에서와 같이 비활당영역에서도 과거에 사용되었거나 삭제된 Pattern이 존재할 수 있다고 생각되어 주어진 data.img를 대상으로 Pattern matching을 시도하였다.

사용한 스크립트는 Gesturecrack을 변형하여 작성하였으며 첨부1과 같다. 해당 스크립트를 통해 기 발견한 070401020508 이외에도 3가지 Pattern을 추가로 획득할 수 있었다. (0001020508, 00010204060708, 000102050807063004)

```
('ffec1e70d113b0b96c7e6cb2b33460e24407a96e', '[0, 1, 2, 5, 8]')  
( '447fd4362cc0b65c1a2aae44316c3e2a33ba4fa9', '[7, 4, 1, 2, 5, 8]')  
( '6a062b9b3452e366407181a1bf92ea73e9ed4c48', '[0, 1, 2, 4, 6, 7, 8]')  
( '77a51db4aca6aa92a43b7e2682d775805547899d', '[0, 1, 2, 5, 8, 7, 6, 3, 4]')
```

[그림 6] 획득한 패턴 크랙

- Password/PIN 복구

PIN과 Password를 복구하기 위해서는 /data/system에 존재하는 “Password.key” 파일을 통해 복구할 수 있다. PIN과 Password 모두 동일 형식으로 저장되며 salt와 password를 concatenation한 값을 MD5와 SHA1 처리한 값을 나타낸다. SHA1(salt|passcode)|MD5(salt|passcode) 와 같이 표현할 수 있다. Salt는 locksettings.db에 존재하는 lockscreen.password_salt의 값을 Hex로 표현한 것을 사용한다.

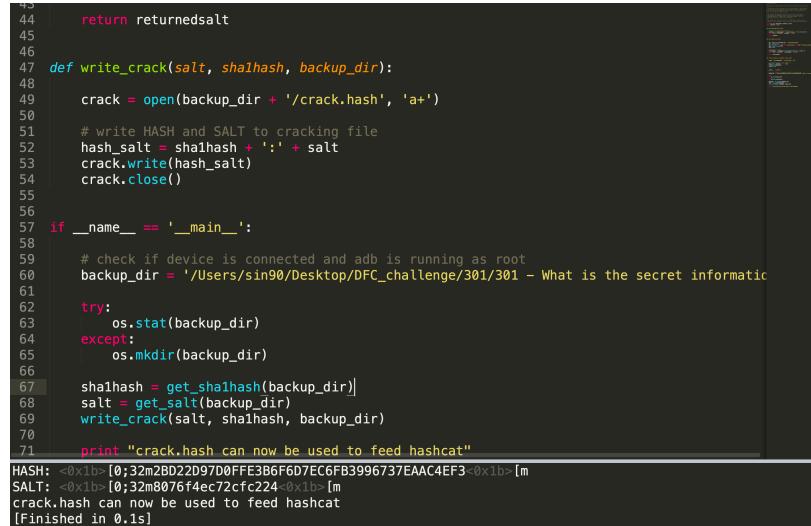
```
(base) sin90@odongbin-ui-MacBookPro 301-Whatshesecretinformation % cat strings.t  
xt | grep lockscreen.password_salt | sort | uniq -c  
79 3lockscreen.password_salt8671682885601775202"  
1 5lockscreen.password_salt-9189888693227240924  
1 lockscreen.password_salt
```

[그림 7] 존재하는 salt 확인

위의 결과와 같이 locksettings.db와 비활당영역에 총 2가지의 salt가 존재함을 확인할 수 있었다.

또한, 정규표현식([A-F0-9]{72})을 이용하여 locksettings.db에 있는 lockscreen.passwordhistory,

password.key, 비활당영역에 존재하는 locksettings.db의 locksreen.passwordhistory를 포함하여 총 6개의 password hash를 얻을 수 있었다. 즉 password hash 6개, salt 2개 총 12개의 조합을 얻을 수 있었다. Salt의 Hex 변환과 Password Hash로부터 Hash 값 추출은 해당 홈페이지에서 제공하는 소스코드²를 이용하였다.



```

43     return returnedsalt
44
45
46
47 def write_crack(salt, sha1hash, backup_dir):
48
49     crack = open(backup_dir + '/crack.hash', 'a+')
50
51     # write HASH and SALT to cracking file
52     hash_salt = sha1hash + ':' + salt
53     crack.write(hash_salt)
54     crack.close()
55
56
57 if __name__ == '__main__':
58
59     # check if device is connected and adb is running as root
60     backup_dir = '/Users/sin90/Desktop/DFC_challenge/301/301 - What is the secret information'
61
62     try:
63         os.stat(backup_dir)
64     except:
65         os.mkdir(backup_dir)
66
67     sha1hash = get_sha1hash(backup_dir)
68     salt = get_salt(backup_dir)
69     write_crack(salt, sha1hash, backup_dir)
70
71 print "crack.hash can now be used to feed hashcat"

```

HASH: <0x1b>[0;32m2BD22D97D0FFE3B6F6D7EC6FB3996737EAAC4EF3<0x1b>[m
SALT: <0x1b>[0;32m8076f4ec72fcf224<0x1b>[m
crack.hash can now be used to feed hashcat
[Finished in 0.1s]

[그림 8] Salt Hex 변환 및 Hash 값 추출

위 그림과 같이 얻은 12개의 조합은 Hashcat을 이용하여 bruteforce를 진행하였다. SHA1보다 MD5가 속도 측면에서 유리할 것이라 판단하여 Password Hash 부분 중에서 SHA1 부분을 제외한 MD5 부분을 추출하여 사용하였다.

6A3C230E790E8E5E51E2580DBC0F9906:785801a3d120ce62
6B365BF036353B7FB5B3E3A275AFC3DF:785801a3d120ce62
A62469279B574C5D6C64D9416ED782BA:785801a3d120ce62
6A3C230E790E8E5E51E2580DBC0F9906:8076f4ec72fcf224
6B365BF036353B7FB5B3E3A275AFC3DF:8076f4ec72fcf224
A62469279B574C5D6C64D9416ED782BA:8076f4ec72fcf224
5FE0E7D01FA73CE99A14E63AA840CAF7:8076f4ec72fcf224
5FE0E7D01FA73CE99A14E63AA840CAF7:785801a3d120ce62
B52BD9B81515F7DB388B0104E221F195:8076f4ec72fcf224
B52BD9B81515F7DB388B0104E221F195:785801a3d120ce62
F2BADF6669159E81476A8147F76435FB:8076f4ec72fcf224
F2BADF6669159E81476A8147F76435FB:785801a3d120ce62

[그림 9] MD5 추출

Bruteforce에 사용된 Masking 조합은 아래와 같다.

- d?d?d?d? : 숫자 4자리
- a?a?a?: 특수문자 포함 4자리
- h?h?h?h?h?h? : [0-9a-f] 6자리
- 1?1?1?1?1?1? : Custom charset [0-9a-z] 7자리

² <https://forensics.spreitzenbarth.de/2015/08/12/breaking-the-screenlock-a-short-update/>

Hashcat을 이용하여 Password가 발견되면, 아래와 같이 Recovered에 나타나게 된다.

```
Session.....: hashcat
Status.....: Running
Hash.Name....: md5($pass.$salt)
Hash.Target...: crack_md5_saltver2.hash
Time.Started.: Mon May 17 18:19:52 2021 (19 mins, 43 secs)
Time.Estimated.: Mon May 17 21:07:46 2021 (2 hours, 28 mins)
Guess.Mask....: ?1?1?1?1?1?1 [7]
Guess.Charset.: -1 ?d?, -2 Undefined, -3 Undefined, -4 Undefined
Guess.Queue....: 1/1 (100.00%)
Speed.#2.....: 6384.2 KH/s (6.60ms) @ Accel:4 Loops:64 Thr:8 Vec:1
Speed.#3.....: 9302.5 KH/s (10.78ms) @ Accel:32 Loops:16 Thr:64 Vec:1
Speed.#*.....: 15686.8 KH/s
Recovered.....: 5/12 (41.67%) Digests, 0/2 (0.00%) Salts
Progress.....: 17242783744/156728328192 (11.00%)
Rejected.....: 0/17242783744 (0.00%)
Restore.Point.: 111104/1679610 (6.61%)
Restore.Sub.#2.: Salt:1 Amplifier:25664-25728 Iteration:0-64
Restore.Sub.#3.: Salt:1 Amplifier:13120-13136 Iteration:0-16
Candidates.#2.: z9ed290 -> 9yibads
Candidates.#3.: gieoza1 -> 62sqd61
6a3c230e790e8e5e51e2580dbc0f9906:8076f4ec72fc224:ninano1
[s]tatus [p]ause [b]ypass [c]heckpoint [q]uit => █
```

[그림 10] Hashcat 사용 화면

2개의 Salt와 Masking 4종류를 이용하여 6개의 Hash로부터 Password를 모두 복구할 수 있다.

```
b52bd9b81515f7db388b0104e221f195:785801a3d120ce62:2222
5fe0e7d01fa73ce99a14e63aa840caf7:785801a3d120ce62:1111
f2badf6669159e81476a8147f76435fb:785801a3d120ce62:a111
a62469279b574c5d6c64d9416ed782ba:8076f4ec72fc224:ace111
6b365bf036353b7fb5b3e3a275afc3df:8076f4ec72fc224:ace222
6a3c230e790e8e5e51e2580dbc0f9906:8076f4ec72fc224:ninano1
```

Android Kitkat에서는 PIN의 경우 숫자 4자리가 가능하고 Password는 숫자4자리 및 숫자로만 이루어진 조합이 불가능하다. 이는 복구된 2222, 1111은 PIN이고, 나머지 ace111, ace222, ninano1은 Password임을 확인할 수 있다.

3. What is the A's secret information? (100 points)

용의자는 Secretbox라는 어플리케이션을 통해 메세지를 암호화하였음. 암호화된 내용을 복호화한 결과는 “Fri 16 May 13:00:55 PM WoderLand”로 파악됨.

용의자가 사용한 어플리케이션 중 secretbox라는 어플리케이션을 발견할 수 있었다. 해당 어플리케이션의 apk 파일을 FTK Imager를 이용하여 추출하고 Android Studio에서 제공하는 Android Virtual Device의 kitkat VM에 설치하여 분석을 진행하였다.



[그림 11] 어플리케이션 동작 화면

Encrypt 버튼을 누르면 Base64 형태의 암호문이 생성되고, SHOW ME THE SECRET이라는 버튼을 누르면 암호문이 복호화되어 나타나는 것을 확인할 수 있었다. 해당 부분을 좀더 명확히 파악하기 위해 JADX를 이용하여 apk decompile 후 소스코드를 확인하였다.

```
public void SetListener() {
    this.showButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View param1View) {
            MainActivity mainActivity = MainActivity.this;
            mainActivity.secret = PreferenceManager.getString(mainActivity.context, "secret");
            MainActivity.this.editTextSecretToken.setText(MainActivity.this.secret);
            String str1 = MainActivity.this.TAG;
            StringBuilder stringBuilder1 = new StringBuilder();
            stringBuilder1.append("secret:");
            stringBuilder1.append(MainActivity.this.secret);
            Log.d(str1, stringBuilder1.toString());
            str1 = EncryptionUtils.decrypt(MainActivity.this.context, MainActivity.this.secret);
            String str2 = MainActivity.this.TAG;
            StringBuilder stringBuilder2 = new StringBuilder();
            stringBuilder2.append(" Decrypted Value :");
            stringBuilder2.append(str1);
            Log.d(str2, stringBuilder2.toString());
        }
    });
}
```

[그림 12] MacinActivity에 존재하는 Decrypt 함수

MainActivity에 Decrypt 함수가 존재하는 것을 확인할 수 있으며 API 18 버전인 현 버전 상에서 암/복호화에 필요한 공개키와 개인키를 사용하기 위해 Android의 Keystore 저장소를 이용하는 것을 확인할 수 있다.

```

private static SecurityKey getSecurityKey(Context paramContext) {
    String str = TAG;
    StringBuilder stringBuilder = new StringBuilder();
    stringBuilder.append("Build.VERSION.SDK_INT:");
    stringBuilder.append(Build.VERSION.SDK_INT);
    Log.d(str, stringBuilder.toString());
    if (Build.VERSION.SDK_INT >= 23) {
        Log.d(str, "M");
        return EncryptionKeyGenerator.generateSecretKey(getKeyStore());
    }
    if (Build.VERSION.SDK_INT >= 18) {
        Log.d(str, "JELLY_BEAN_MR2");
        return EncryptionKeyGenerator.generateKeyPairPreM(paramContext, getKeyStore());
    }
    Log.d(str, "generateSecretKeyPre18");
    return EncryptionKeyGenerator.generateSecretKeyPre18(paramContext);
}

```

[그림 13] KeyStore 저장소 활용

데이터를 실제 암호화하기 위해서 RSA/ECB/PKCS1Padding 기법을 이용한다는 것도 알 수 있으며 Base64 encode와 Decode시 parameter 값을 통해 URL_SAFE(8)기법을 이용한다는 것도 확인할 수 있었다.

```

private Cipher getCipher(int paramInt) throws GeneralSecurityException {
    if (Build.VERSION.SDK_INT >= 23) {
        Cipher cipher1 = Cipher.getInstance("AES/GCM/NoPadding");
        cipher1.init(paramInt, this.secretKey, new GCMParameterSpec(128, "AES/GCM/NoPadding".getBytes());
        return cipher1;
    }
    if (Build.VERSION.SDK_INT >= 18) {
        PrivateKey privateKey;
        Cipher cipher1 = Cipher.getInstance("RSA/ECB/PKCS1Padding");
        if (paramInt == 2) {
            PublicKey publicKey = this.keyPair.getPublic();
        } else {
            privateKey = this.keyPair.getPrivate();
        }
        cipher1.init(paramInt, privateKey);
        return cipher1;
    }
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    cipher.init(paramInt, this.secretKey, new IvParameterSpec(new byte[cipher.getBlockSize()]));
    return cipher;
}

String decrypt(String paramString) {
    if (paramString == null)
        return null;
    try {
        return new String(getCipher(2).doFinal(Base64.decode(paramString, 8)));
    } catch (GeneralSecurityException generalSecurityException) {
        Log.e(this.TAG, String.valueOf(generalSecurityException));
        return null;
    }
}

```

[그림 14] 암호화 및 인코딩 방식 파악

또한 암호화한 값은 secret_preference.xml에 저장하고 복호화 시 가져온다는 것을 확인할 수 있었다.

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="[Secret]">zAkK8mNxImFfBdNGgrz6F7iP61F3wS7haKE_cjiaXm5lL-tuSLxVo_DooAv1azl5G
gShfHX7TfWaaIkLTQY-Cvflyk0QcKnaBZrHgKP3tZX-XXFO-fViRbgtBSwhHF_6dEzFcyjqeZqIDs24khIe4UQT
Kw0SsJmVVcKfxa4jH7VqyU-dkyP9kh-suU9X3pUChEQb8X3IJwibP9GgN7c73UXd50vA_SMnLMGYll_iePT2vn4
SUi0sBRyi0SGg8VuGVDtob4IRLHiQ3Ivzqm2A4kRk_ooybVDJvRH83KBmsbm7kIunJnGpexy2BHxIQm3fUYFyle
QHE7IX1PmUKnP8qA==
    </string>
</map>
```

[그림 15] secret_preference.xml에 저장된 암호문

즉 우리가 원하는 secret_preference.xml 에 저장된 Base64 URL safe encoding 된 암호문을 Keystore 로부터 복호화에 필요한 key 를 추출하여 RSA/ECB/PKCS1Padding 로 복호화를 진행해야함을 알 수 있다.

Base64 URL safe encoding 의 경우 URL 에서 사용할 수 있기 위해 문자를 치환한 형식이다. 이를 우리가 사용하는 일반적인 base64 로 변환하기 위해서 -와 _를 +와 /로 치환해주어야한다. 치환된 base64 는 다음과 같다.

```
zAkK8mNxImFfBdNGgrz6F7iP61F3wS7haKE/cjiaXm5lL+tuSLxVo/DooAv1azl5GgShfHX7TfWaalkLTQY+Cvfl
ykOQcKnaBZrHgKP3tZX+XXFO+fViRbgtBSwhHF/6dEzFcyjqeZqIDs24khle4UQTKw0SsJmVVcKfxa4jH7VqyU
+dkyP9kh+suU9X3pUChEQb8X3IJwibP9GgN7c73UXd50vA/SMnLMGYll/iePT2vn4SUi0sBRyi0SGg8VuGVDt
ob4IRLHiQ3Ivzqm2A4kRk/ooybVDJvRH83KBmsbm7kIunJnGpexy2BHxIQm3fUYFyleQHE7IX1PmUKnP8qA=
=xq
```

이후에는 Keystore 로 부터 복호화에 필요한 key 를 추출해야한다. Android Kitkat 에서는 Keystore 에 사용된 공개키(CERT)와 개인키(PKEY)를 /data/misc/keystore/user_0/ 에 보관한다. 이 Key 값을 구하기 위해서는 해당 폴더에 존재하는 .masterkey 와 현재 사용하고 있는 screen lock password 가 요구된다. 해당 파일들의 구조는 다음³에서 확인할 수 있다.

현재 사용되고 있는 화면 잠금 방법은 Pattern 이므로 Pattern 을 Password 와 같이 입력해 주어야한다. Android Keystore 를 구현하기 위한 소스코드 상에서도 pw 라는 변수를 그대로 입력하는 것을 확인할 수 있었다.⁴ 이는 Pattern 의 경우 String 이 아닌 Hex value 그대로를 복호화하는 함수의 Parameter 로 사용해야하는 것을 의미한다. Android kitkat 의 경우 Password 변경되면 관련 Keystore Key 들도 삭제되므로, 가장 최신인 070401020508 이 현재 Keystore 생성에 관여하고 있음을 알 수 있다.

.masterkey 와 password 를 이용하여 Keystore 의 Key 를 복구하는 도구로 대표적인 것인 android-ks-decryptor 이다. 해당 도구는 keystore-decryptor 라는 도구의 python 포팅 버전이다. 그러나 이 도구들 모두 PIN / Password만 입력 가능하고 Pattern은 불가능하다. 이를 해결하기 위해 소스코드 수정이 필요하다. 수정된 소스코드(중요부분)은 첨부 2 로 확인가능하다.

³ <https://cozyu.tistory.com/329>

⁴

<https://android.googlesource.com/platform/system/security/+/bb9f4392c2f1b11be3acdc1737828274ff1ec55b/keystore/keystore.cpp>

스크립트를 통해 FTK Imager로 추출한 /data/misc/keystore/user_0/ 상에 존재하는 .masterkey, 10055_USRPKEY_KEY_ALIAS, 10055_USRCERT_KEY_ALIAS 파일을 복구하여 아래와 같은 결과를 획득할 수 있다.

```
(base) sin90@odongbin-ui-MacBookPro user_0 % python android-ks-decryptor_pattern
tohex.py --master-key masterkey_tirmed --password 741258 10055_USRPKEY_KEY_ALIAS
_trimed
length : 84
description : None
version : 2
type : 2
flags : 1
info : 16
blob : <_io.BufferedReader name='masterkey_tirmed'>
salt : 0cdfde73ba894a0fe3ddef2a5bc54ffe
['07', '04', '01', '02', '05', '08']
b'\x07\x04\x01\x02\x05\x08'
48
c455f46235d95418b51741ad9c4a6997
c455f46235d95418b51741ad9c4a6997
1232
355459135a1528c1b04678f7237ce89f
355459135a1528c1b04678f7237ce89f
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEaZ84C5dxD0704VLu++j4+ZzNNUEYr08is1ZFACvSB01Yri8
tVAOpoeBAhLq1wI7sRM2NH7dWV3mzahpIVJGwdsFtpfLktbTEoAgFxTT8k9Z/
ZJ2PMryhm6VyxPb2Agrngp77+wMHiaBhcPf2uu+18EBrrEpuHkIUdynWiy3Eyc
KEP9Fva5vYFwsWWFsVA907KBVCfIe4FKUd0fYkeC5+kwVh9MtQQQ29K0suCDCrp0
NUfv5n93ShQaCMpyyRyQykv4LTK7nIDAx/NVuRQWwbReQhB8FXDLyIMSp/1cf
AJAxuSwyzYL+8mjF12ThAr91YBAza2qvzywZwIDAQABoIBACEx8oqPffn6gpae
iyPKaQmJS290vh2tt8wIsPMBMQK07Xbutc5s5Ho+ok7YmGxeRnsH2pM4L+gX9lXI
BNhIJ1420eqjIEfjhH/cqZA619kh1rszB5SmCqr7MQGxEy0B/Kfk4AigpvX5DNmW
U+tAb9gD0qVhQoiKaz5jVVLvm+KC/mGTF+P5RnDileHka842+5Y1wCAFVDYwx8MI
5LIRQS3msndw+n8glRbvsJ4IOBFzW2TFjkxrxa074OsR5hk1POKnvh78HZFrqq6Q
AyLLhVhRuWXQsfze08JankC8uSMTGOfh2V/D1XmgQwp70eKFp9jwEL1bWTAdnct
C1zk6EcgYEAT7EJ3csDsDjAscYcf1FWMMLyxEDdJVEASf1vSZBep6owQZrYJRY/
8jo02EVH4KgsX90GRW7zg0E++vP8GCDSF+EacAVDSTpyUPD+wR3LISIWFfbxDiA
kPX4rOy5+Xjrg9qo2WzzfC13WteWjWwz/zEktU5KToy60Dg3cPfrECgYEAS4Srm
zp9cxWloLDABwwcyWHC/ZgIy7u4oERsq90y1j2TeVcnj1kMc2GxibKyZ+0abk1
10vyLLc4kb1z0SAaDV6EAGVoEonaAAkGYbbw1iu8Dpn8Zhe1XoTTGC/DBI1cXbkq
GSmrnmY6HUWB+XIoY5R+s+yofAgcjFMTF9qB1pcCgYbtGPg91laqKn1LY2FQVLY/
Kib9XPpQnFGuXAIFYVRESjOP0IaD4XXFBfgi2IMsOLVXgyHdODHcn2lsyvPTVHyP
qFPj/Joom40kwTTY6Adibb222+k4eSEjAdsvHzmehvf15QU35IqYp+J5GTPpVF
jic5gG3QM1KIkvk3YOhX0QKBgQDKOGd1zfZBbXsbRDow14PFneynRo5LuyPMysN5
OCWPE17D7E8zIi8Dx8NWHE6QU/+sFJReF1E2C48IhKa4xTxgj+PphymqjJd2tKRA
mbYOAtUwH+doGzn+ezfCxk8qeCUY7mFQ8FxzuOea1bioL3ZQQRwvdHnV1WRT2+
j1VTHQKBgDmjUwTY3Ik1EoSBoCsf0Ie70G4pjYUcgNDKL7VLocgDyZ1Y1zf/gDq9
CSSDU++KL3iM8qV9jlsitaPHvl1DqT/VVHrb/kVMBF+2PtWP1EUgi50ibxqErSqp
AtBzJAfhqc7b7uHSWhURWvW2AvQdhDwMBBICzIadaIlsoJbHmnL/
-----END RSA PRIVATE KEY-----
(base) sin90@odongbin-ui-MacBookPro user_0 %
```

[그림 16] RSA Key 덤프

결과로 나오는 RSA Key는 --dump-pem Option을 통해 덤프 가능하였다.

이후에는 암호화 복호화 과정에 사용되는 Key를 알아보기 위해 다시 동적 분석을 진행하였다. Frida를 이용하여 진행하였으며 Android에서 자주 사용되는 암호화 라이브러리 함수를 Hooking 하도록 정의된 라이브러리⁵를 활용하였다. 그 중 위의 소스코드 상에서도 확인할 수 있는 cipher 함수를 대상으로 tracer-cipher.js를 injection 및 후킹을 진행하였다. 이후 암복호화하여 다음의 결과를 얻을 수 있었다.

⁵ <https://github.com/FSecureLABS/android-keystore-audit>

```

Cipher hooks loaded!
[Android Emulator: 5554:com.example.secretbox] -> [Cipher.getInstance(): type: RSA/ECB/PKCS1Padding
[Cipher.getInstance(): cipherObj: javax.crypto.Cipher@9d0b3400
[Cipher.init()]: mode: Decrypt mode, secretKey: com.android.org.conscrypt.OpenSSLRSApublicKey , cipherObj: javax.crypto.Cipher@9d0b3400
[Cipher.init()]: mode: Decrypt mode, secretKey: com.android.org.conscrypt.OpenSSLRSApublicKey secureRandom:java.security.SecureRandom@9d0706e8 , cipherObj: javax.crypto.Cipher@9d0b3400
[Cipher.doFinal2()]: cipherObj: javax.crypto.Cipher@9d0b3400
In buffer:

Offset 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F .....QX\.%m1...
00000000 2C 87 F7 A9 8E 19 51 58 5C B3 25 6D 31 96 BA 10 .....#...y8...0..
00000010 A2 B4 14 AA B1 23 A7 F2 1F 79 38 A1 DB 30 ED CF .....p.M.6...).*.].
00000020 5F 70 AC 4D AA 36 C4 FE B3 29 F9 2A 13 F2 5D 13 .....q...zW...'.nCo
00000030 5D 36 68 CC A8 72 57 8C 33 FB 18 2D 2B 8C 24 16h..rvW.3.-+.$
00000040 9A BC A6 FA 7D D6 BF FE 3A FE 80 AC 32 B1 11 6A .....p.....2.j
00000050 26 D0 1E D6 FE 93 FB 16 AF 73 92 62 98 C7 73 F7 &.....s.b.s.
00000060 12 5B FD A5 5C 4C 1A 28 9F AA 92 55 D1 A1 64 43 [...]L. ....U..DC
00000070 9C 14 71 AA AC 98 7A 57 0A D6 27 E6 98 6E 43 6F ..q...zW...'.nCo
00000080 99 92 F3 B8 A3 3C BB AB 0F 54 72 AC 91 FA 7F A3 .....<...Tr...
00000090 93 E1 CA 9A C4 45 6E 08 F7 4B 33 6A F7 AE 41 97 .....En..K3j..A.
000000A0 53 C0 6F 6E BF D6 E9 37 A7 D5 7B 7A B6 6F F1 31 S.on....J{.o.1
000000B0 EF F1 59 D5 EA B5 E0 A5 A8 0E 2D 8C 3B 7D 92 E1 ..Y.....-.;..
000000C0 F3 2B 4E 63 71 9E 31 04 09 DB 7C F6 8C 95 2E 64 .+Ncq.1...|....d
000000D0 9C 58 4F 61 B8 35 D7 0F 57 3F B3 B9 A0 1D F3 58 .XoA..5..W?....X
000000E0 5B 0E 84 0B 83 C8 F7 87 6A 6B 74 5A D0 08 A3 11 [.....jktZ.....
Result:
Offset 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F testtest
00000000 74 65 73 74 74 65 73 74
[Cipher.getInstance(): type: RSA/ECB/PKCS1Padding
[Cipher.getInstance(): cipherObj: javax.crypto.Cipher@9d190ea8
[Cipher.init()]: mode: Encrypt mode, secretKey: com.android.org.conscrypt.OpenSSLRSAprivateKey , cipherObj: javax.crypto.Cipher@9d190ea8
[Cipher.init()]: mode: Encrypt mode, secretKey: com.android.org.conscrypt.OpenSSLRSAprivateKey secureRandom:java.security.SecureRandom@9d0706e8 , cipherObj: javax.crypto.Cipher@9d190ea8
[Cipher.doFinal2()]: cipherObj: javax.crypto.Cipher@9d190ea8
In buffer:

Offset 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F testtest
00000000 74 65 73 74 74 65 73 74
Result:
Offset 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F .....QX\.%m1...
00000000 2C 87 F7 A9 8E 19 51 58 5C B3 25 6D 31 96 BA 10 .....#...y8...0..
00000010 A2 B4 14 AA B1 23 A7 F2 1F 79 38 A1 DB 30 ED CF .....p.M.6...).*.].
00000020 5F 70 AC 4D AA 36 C4 FE B3 29 F9 2A 13 F2 5D 13 .....q...zW...'.nCo
00000030 5D 36 68 CC A8 72 57 8C 33 FB 18 2D 2B 8C 24 16h..rvW.3.-+.$
00000040 9A BC A6 FA 7D D6 BF FE 3A FE 80 AC 32 B1 11 6A .....p.....2.j
00000050 26 D0 1E D6 FE 93 FB 16 AF 73 92 62 98 C7 73 F7 &.....s.b.s.
00000060 12 5B FD A5 5C 4C 1A 28 9F AA 92 55 D1 A1 64 43 [...]L. ....U..DC
00000070 9C 14 71 AA AC 98 7A 57 0A D6 27 E6 98 6E 43 6F ..q...zW...'.nCo
00000080 99 92 F3 B8 A3 3C BB AB 0F 54 72 AC 91 FA 7F A3 .....<...Tr...
00000090 93 E1 CA 9A C4 45 6E 08 F7 4B 33 6A F7 AE 41 97 .....En..K3j..A.
000000A0 53 C0 6F 6E BF D6 E9 37 A7 D5 4A 7B A6 6F F1 31 S.on....J{.o.1
000000B0 EF F1 59 D5 EA B5 E0 A5 A8 0E 2D 8C 3B 7D 92 E1 ..Y.....-.;..
000000C0 F3 2B 4E 63 71 9E 31 04 09 DB 7C F6 8C 95 2E 64 .+Ncq.1...|....d
000000D0 9C 58 4F 61 B8 35 D7 0F 57 3F B3 B9 A0 1D F3 58 .XoA..5..W?....X
000000E0 5B 0E 84 0B 83 C8 F7 87 6A 6B 74 5A D0 08 A3 11 [.....jktZ.....

```

[그림 17] FRIDA를 통한 Hooking

Testtest라는 문자열을 암호화할 때는 “개인키”를 사용하고 복호화할 때는 “공개키”를 사용함을 알 수 있었다. 현재 필요한 작업은 복호화이므로, 공개키가 필요하다는 것을 알 수 있다. 위와 동일한 방식으로 pattern 입력이 가능하도록 된 스크립트를 이용, 10055_USRCERT_KEY_ALIAS의 RSA 공개키를 얻을 수 있었다.

```

-----BEGIN CERTIFICATE-----
MIICotCCAYmgAwIBAgIBCAjANBgkqhkiG9w0BAQsFADAMRIwEAYDVQQDDAALLRvlf
QUxJ0VMwIhCNMjEwNDE2MDUxMjU1WhcNMjIwNDE2MDUxMjU1WjAUERIwEAYDVQD
DALRvlfOUxJ0VmwggeIMAOGCSqS1b3DQEBAQUAA4IBDwAwggEKAoIBAQDPzgLL
3EM7s7UhU776Pj5m01QRis7KyVvUkBUK91GjUtiuLy1UA6mmh4EC EurXjuxEzY
0ft1XebNqGkhkZa92wZ2l8u51tmSgCAXFPyT1n9kn8yvKGbpXekHYCueC
nvv7AwchpuFynt/a676XwQqusS6m4eQhR3KdaLlcTjyQ0/0W9rm9gXcxZYwxUD07
soFUJ8h7UpR3R91RAh6L6TBW0h1yBB0d0rSy4IMKunQ1R+mf3dKFBoIyinLLFHJ1
rkS/gtMrudAgMDH81W5FBZ2f5CEHwVmVigKn+vWUAkG5LDLNg7wyMwZ0EC
v2PgDTNrag/PLBnAgMBAEwDQYJKoZIhvchNAQELBQAdggEBAAnteE000maEG7I
gh511j8500xt0VBBGma08HfkhtJGz70zHAn6E9gsAmwHKKVV4kJA+8DWU16FB1
IfvZiuNb/YSl19z/Y97HWPVcGZt3PeAE0zsrAoKoHjb0nTeEyjEfoPyG7F0kRS
HOYLz1+7LWttr6/GxPRkyZIRwG1U39ow2lu+jmK1518ABF1m+4Hoas5Rw0m0sD00
jEqf+FQH2L5kFbBFWgG/Tad0QRlcM6/7rR3rXfQ5XjACLzrp54/2LGUHrKge9vgr
IuFCSDVrzrUQ3hra1obqCousi7ae7wtHSL4icdOLWG1s75/H83dwbmwdzfULLL4
hy44Kow=
-----END CERTIFICATE-----

```

[그림 18] 공개키 획득

RSA/ECB/PKCS1Padding 을 decrypt 하기 위해서는 online site⁶를 이용하였다. 해당 사이트에 Public Key, Base64 message 를 넣고 최종적으로 아래와 같은 메세지를 얻을 수 있었다.

⁶ <https://8gwifi.org/rsafuctions.jsp>

Public Key	Private Key
	<pre>-----BEGIN CERTIFICATE----- MIICoTCCAYmgAwIBAgIBCjANBgkahkiG9w 0BAQsFADAUMRIwEAYDVQQDDAIlRVlf QUxJQVMwHhcNMiEwNDE2MDUxMiU1Wh cNMiIwNDE2MDUxMiU1WjAUMRIwEAYDV QQD DAILRVlfQUxJQVMwgqEiMA0GCSqGSIb3D QEBAQUAA4IBDwAwggEKAoIBAQDPzgLI 3EM7s7hUu776Pi5nM01QRis7yKyVkJUBUK 9IGiUtiuLy1UA6mmh4EC EurXAjuxEzY</pre>
ClearText Message	output
<pre>s24knle4UQTkwUssJmVVcKtxa4JH/VqvU+ dkyP9kh+suU9X3pUChEQb8X3IJwibP9Gg N7c73UXd5OvA/SMnLMGYII/ePT2vn4SUi0 sBRyi0SGg8VuGVDtob4IRLHiQ3lvzam2A4 kRk/ooybVDJvRH83KBmsbm7klunJnGpexy 2BHXIQm3fUYFyleQHE7IX1PmUKnP8qA==</pre>	<pre>Fri 16 May 2021 13:00:55 PM, WoderLand</pre>

[그림 19] 암호화 메시지 복호화 결과

“Fri 16 May 13:00:55 PM WoderLand” 를 숨기고자 한 것을 확인하였다.

첨부1.

```
import sqlite3 as lite
import re
db
lite.connect('/Users/sin90/Desktop/DFC_challenge/301/GestureCrack/rainbow.db') =
cur = db.cursor()
rows = cur.fetchall()
f = open('data.img','rb')
data= f.read()
import codecs
from tqdm import tqdm
data = codecs.encode(data, 'hex_codec').decode()
def detector(row,data) :
    p=re.findall(row[0],data)
    if len(p) > 0 :
        return row
import parmap
result = parmap.map(detector, rows, data, pm_pbar=True , pm_processes = 8)
for i in result :
    if i != None :
        print (i)
```

첨부2.

```
encrypted, salt = b[:-ksb.info], b[-SALT_SIZE:]
print("salt : %s" %salt.hex() )
password =list(password)
new_password = []
for i in password :
    temp= '0'+i
    new_password.append(temp)
print(new_password)
new_password = ' '.join(new_password)
new_password = bytes.fromhex(new_password)
print(new_password)
kek = ksb.generate_kek(new_password, salt)
decrypted = BytesIO(ksb.decrypt(iv, encrypted, kek))
digest = decrypted.read(MD5_DIGEST_LENGTH).hex()
digest_calculated = md5(decrypted.read()).hexdigest()
assert digest == digest_calculated
```