

305 – Crack the app

Team Information

Team Name : DogeCoin

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

Email Address : dfc-dogecoin@naver.com

Instructions

Description You just seized the suspect's cell phone. To analyze the cell phone, you must login to unlock the private drive application. The suspect refused to provide the passcode, so you need to crack the password through reverse engineering.

Target	Hash (SHA256)
PrivateDrive.apk	7792591f3ca703d113e65030d3f1041bc18ec6e2 26f8fc98d2504d03485e616b

Questions

1. Suggest the appropriate method to crack the passcode. If you have implemented a program to solve the problem, submit the source code together.
2. After unlocking the passcode, take screenshots with the 'first password' and OTP code.

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

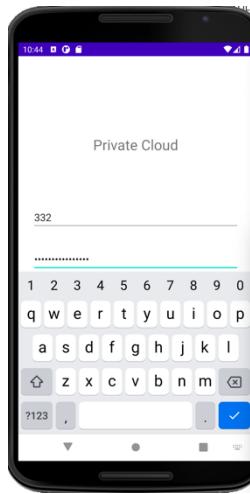
Name:	JADX	Publisher:	skylot
Version:	1.2.0		
URL:	https://github.com/skylot/jadx		

Name:	Android Studio	Publisher:	Google Developers
Version:	202.7351085		
URL:	https://developer.android.com/studio		

Step-by-step methodology:

1. Suggest the appropriate method to crack the passcode. If you have implemented a program to solve the problem, submit the source code together.

분석을 위해 Android Studio를 이용하여 문제에서 주어진 PrivateDrive.apk를 설치 후 구동하였다. 그 결과는 아래의 [그림 1]과 같이 Login에 필요한 Username과 Password를 요구하고 있는 것을 확인할 수 있었다.



[그림 1] PrivateDrive 구동

로그인 로직을 이해하기 위해 JADX라는 도구를 이용, APK decompile을 진행하였다. APK Decompile을 통해 아래의 [그림 2]와 같이 MainActivity로부터 LoginResult.Login 함수를 호출하고 있음을 확인할 수 있다.

```
public class MainActivity extends AppCompatActivity {
    /* access modifiers changed from: protected */
    @Override // androidx.activity.ComponentActivity, androidx.core.app.ComponentActivity, androidx.fragment.app.FragmentActivity
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(C0605R.layout.activity_main);
        final EditText usernameEditText = (EditText) findViewById(C0605R.C0608id.username);
        final EditText passwordEditText = (EditText) findViewById(C0605R.C0608id.password);
        final ProgressBar loadingProgressBar = (ProgressBar) findViewById(C0605R.C0608id.loading);
        (Button) findViewById(C0605R.C0608id.login).setOnClickListener(new View.OnClickListener() {
            /* class com.example.privatedrive.MainActivity$View$OnClickListenerC06041 */
        });
        public void onClick(View v) {
            loadingProgressBar.setVisibility(0);
            boolean result = false;
            try {
                result = LoginResult.login(usernameEditText.getText().toString(), passwordEditText.getText().toString(), MainActivity.this.getString(C0605R.string.password_key), MainActivity.this);
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            }
            if (result) {
                Intent intent = new Intent(MainActivity.this.getApplicationContext(), SubActivity.class);
                intent.putExtra("username", usernameEditText.getText().toString());
                intent.putExtra("password", passwordEditText.getText().toString());
                MainActivity.this.startActivity(intent);
                return;
            }
            Toast.makeText(MainActivity.this.getApplicationContext(), "Login Failed", 1).show();
            loadingProgressBar.setVisibility(4);
        }
    }
}
```

[그림 2] MainActivity함수

LoginResult.Login 함수 구조는 아래와 같다.

```
package com.example.privatedrive;

import android.content.Context;
import java.io.FileNotFoundException;
import kotlin.UByte;

public class LoginResult {
    public static boolean login(String username, String password, String pw_key, Context applicationContext) throws FileNotFoundException {
        int pw_len = password.length();
        byte[] keyArray = new byte[pw_len];
        byte[] keyArray1 = password.substring(0, pw_len / 2).getBytes();
        byte[] keyArray2 = password.substring(pw_len / 2, pw_len).getBytes();
        byte[] keyChar = {16, 5, 18, 13, 10, 26, 61, 5, 82, 3, 1, 24, 19, 36, 108, 15};
        for (int i = 0; i < keyArray1.length; i++) {
            keyArray1[i] = (byte) (keyArray1[i] ^ keyChar[i]);
        }
        for (int i2 = 0; i2 < keyArray2.length; i2++) {
            keyArray2[i2] = (byte) (keyArray2[i2] ^ keyChar[i2]);
        }
        System.arraycopy(keyArray2, 0, keyArray, 0, keyArray2.length);
        System.arraycopy(keyArray1, 0, keyArray, keyArray2.length, keyArray1.length);
        return byteArrayToHex(keyArray).equals(pw_key);
    }

    public static String byteArrayToHex(byte[] a) {
        StringBuilder sb = new StringBuilder();
        int length = a.length;
        for (int i = 0; i < length; i++) {
            sb.append(String.format("%02x", Integer.valueOf(a[i] & UByte.MAX_VALUE)));
        }
        return sb.toString();
    }
}
```

[그림 3] LoginResult.Login 함수

LoginResult.Login 함수를 보면 실질적으로 함수의 Parameter로 사용되는 “username”은 내부에서 아무런 역할을 하지 않고 있음을 확인할 수 있다. password의 경우 자체 로직에 의하여 변형된 값을 사전 정의된 값과 비교한다. 비교한 결과를 Return하여 최종적으로 Login 여부를 결정하게 된다. 자체 로직은 다음과 같다.

- 1. 입력된 password를 반으로 나눈다. [**Password = (Keyarray[1] || Keyarray[2])**]
- 2. 반으로 나누어진 Keyarray들을 사전에 정의된 KeyChar 배열의 요소들의 순서에 맞추어 XOR 연산을 수행한다. **Keyarray[i][k]= Keyarray[i][k] ^ KeyChar[k]**
- 3. XOR 된 Keyarray를 역순으로 조합한다. [**Result =(Keyarray[2] || Keyarray[1])**]

위의 로직에 따라 변경된 Result 값을 소스코드 상 “pw_key”에 들어가는 값과 비교하게 된다. Passcode를 crack하기 위해 제일 먼저 해야할 것은 역연산이 가능한지 여부이다. 만약 내부 로직에 역연산이 불가능한 Hash Function 등의 사용이 존재하면 Rainbow Table과 같은 것을 활용하여 BruteForce를 진행하여야 한다. 그러나 위의 로직의 경우, 전치와 XOR 연산 밖에 이루어지지 않아 쉽게 역연산을 위한 로직을 구성할 수 있다. 역연산을 위한 로직은 다음과 같다. 특히 XOR의 경우 XOR 된 값에 다시 같은 숫자로 XOR를 할 경우 원래 값을 도출해 낼 수 있다는 특징이 있다.

- 1. 입력된 password를 반으로 나눈다 [**Result = (Keyarray[1] || Keyarray[2])**]
- 2. 반으로 나누어진 Keyarray들을 사전에 정의된 KeyChar 배열의 요소들의 순서에 맞추어 XOR 연산을 수행한다 **Keyarray[i][k]= Keyarray[i][k] ^ KeyChar[k]**
- 3. XOR된 Keyarray를 역순으로 조합한다 [**Original =(Keyarray[2] || Keyarray[1])**]

위의 로직에 근거하여 Input 값으로 사용되는 pw_key를 찾아내면, 역연산을 수행할 수 있다.

```

package com.example.privatedrive;

import android.content.Context;
import java.io.FileNotFoundException;
import kotlin.UByte;

public class LoginResult {
    public static boolean login(String username, String password, String pw_key, Context applicationContext) throws FileNotFoundException {
        byte[] keyArray = new byte[pw_len];
        byte[] keyArray1 = password.substring(0, pw_len / 2).getBytes();
        byte[] keyArray2 = password.substring(pw_len / 2, pw_len).getBytes();
        byte[] keyChar = {16, 5, 18, 13, 10, 26, 61, 5, 82, 3, 1, 24, 19, 36, 108, 15};
        for (int i = 0; i < keyArray1.length; i++) {
            keyArray1[i] = (byte) (keyArray1[i] ^ keyChar[i]);
        }
        for (int i2 = 0; i2 < keyArray2.length; i2++) {
            keyArray2[i2] = (byte) (keyArray2[i2] ^ keyChar[i2]);
        }
        System.arraycopy(keyArray2, 0, keyArray, 0, keyArray2.length);
        System.arraycopy(keyArray1, 0, keyArray, keyArray2.length, keyArray1.length);
        return byteArrayToHex(keyArray).equals(pw_key);
    }
}

public class MainActivity extends AppCompatActivity {
    /* access modifiers changed from: protected */
    @Override // androidx.activity.ComponentActivity, androidx.core.app.ComponentActivity, androidx.fragment.app.FragmentActivity
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(C0605R.layout.activity_main);
        final EditText usernameEditText = (EditText) findViewById(C0605R.C0608Id.username);
        final EditText passwordEditText = (EditText) findViewById(C0605R.C0608Id.password);
        final ProgressBar loadingProgressBar = (ProgressBar) findViewById(C0605R.C0608Id.loading);
        ((Button) findViewById(C0605R.C0608Id.login)).setOnClickListener(new View.OnClickListener() {
            /* class com.example.privatedrive.MainActivity */
            public void onClick(View v) {
                loadingProgressBar.setVisibility(0);
                boolean result = false;
                try {
                    result = LoginResult.login(usernameEditText.getText().toString(), passwordEditText.getText().toString());
                } catch (FileNotFoundException e) {
                    e.printStackTrace();
                }
                if (result) {
                    Intent intent = new Intent(MainActivity.this.getApplicationContext(), SubActivity.class);
                    intent.putExtra("username", usernameEditText.getText().toString());
                    intent.putExtra("password", passwordEditText.getText().toString());
                    MainActivity.this.startActivity(intent);
                }
            }
        });
        Toast.makeText(MainActivity.this.getApplicationContext(), "Login Failed", 1).show();
        loadingProgressBar.setVisibility(4);
    }
}

```

<string name="navigation_drawer_open">Open navigation drawer</string>
 <string name="otp_password">OTPPassword</string>
 <string name="password_key">283d233e382c1e215a6a7c6c7e725c6b</string>
 <string name="password_toggle_content_description">Show password</string>

[그림 4] Value Compare Logic

위의 그림과 같이 “pw_key”에 해당하는 키를 /PrivateDrive/resources/res/values/strings.xml에서 발견할 수 있었다. APK 파일의 경우 PKZIP의 구조를 가지고 있기 때문에 압축해제 프로그램으로 압축해제가 가능하다. 역연산에 사용한 Source Code는 아래 [첨부 1]로 확인할 수 있다.

```

password = [0x20,0x3d,0x23,0x3e, 0x38,0x2c, 0x1e, 0x21, 0x5a,0x6a,0x7c,0x6c,0x7e,0x72,0x5c,0x6b]
keyarray = [16, 5, 18, 13, 10, 26, 61, 5, 82, 3, 1, 24, 19, 36, 108, 15]

password_array1 = password[0:len(password)/2]
password_array2= password[len(password)/2:]

recovered_array2 = []
for k, i in enumerate(password_array1) :
    temp = i^keyarray[k]
    recovered_array2.append(chr(temp))

recovered_array1 = []
for k, i in enumerate(password_array2) :
    temp = i^keyarray[k]
    recovered_array1.append(chr(temp))

print(''.join(recovered_array1) + ''.join(recovered_array2))

```

[첨부 1] 역연산 소스코드 (Python)

위의 Source Code를 통해 아래의 그림과 같은 결과를 얻었다. Passcode는 “Jonathan081326##”이다.

```

1 password = [0x20,0x3d,0x23,0x3e, 0x38,0x2c, 0x1e, 0x21, 0x5a,0x6a,0x7c,0x6c,0x7e,0x72,0x5c,0x6b]
2 keyarray = [16, 5, 18, 13, 10, 26, 61, 5, 82, 3, 1, 24, 19, 36, 108, 15]
3
4 password_array1 = password[0:len(password)/2]
5 password_array2= password[len(password)/2:]
6
7 recovered_array2 = []
8 for k, i in enumerate(password_array1) :
9     temp =i^keyarray[k]
10    recovered_array2.append(chr(temp))
11
12 recovered_array1 = []
13 for k, i in enumerate(password_array2) :
14     temp =i^keyarray[k]
15    recovered_array1.append(chr(temp))
16
17
18 print(''.join(recovered_array1) + ''.join(recovered_array2))
19

```

Jonathan081326#\$
[Finished in 0.1s]

[그림 5] 1차 비밀번호 획득

Username에 해당하는 부분을 임의의 문자를 넣어주고 Password 부분을 Jonathan081326#\$를 입력하면 다음과 같은 2-factor auth code OTP를 입력하라는 화면을 확인할 수 있다.

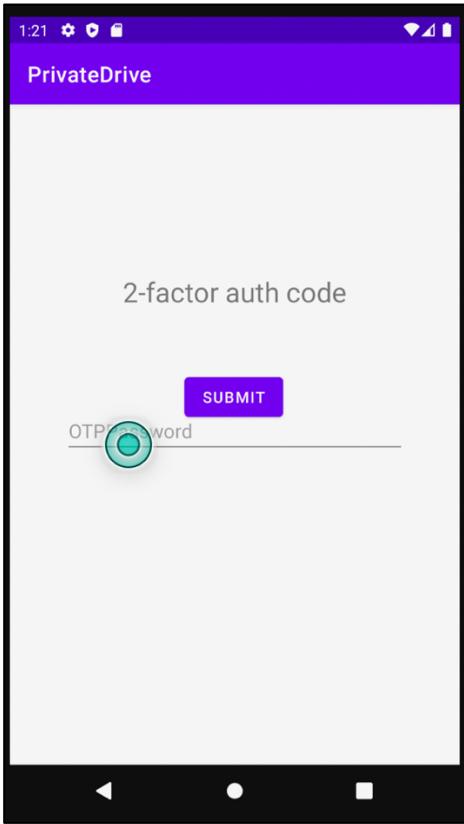


그림 6. 2-factor 관련 입력 요구

관련된 부분의 APK Decompile 된 부분은 아래의 그림과 같이 SubActivity에서 확인할 수 있다.

```

public class SubActivity extends AppCompatActivity {
    /*@Override // androidx.activity.ComponentActivity, androidx.core.app.ComponentActivity, androidx.fragment.app.FragmentActivity
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(C0605R.layout.activity_sub);
        getIntent().getStringExtra("username");
        final EditText OTPEditText = findViewById(C0605R.C0608Id.otppassword);
        (Button) findViewById(C0605R.C0608Id.login).setOnClickListener(new View.OnClickListener() {
            /* class com.example.privatedrive.SubActivity$View$OnClickListener@06121 */
            public void onClick(View v) {
                try {
                    if (OTPEditText.getText().toString().equals(SubActivity.this.getString(C0605R.string.password_key).substring(15,toUpperCase())))) {
                        Toast.makeText(SubActivity.this.getApplicationContext(), "OTP Auth Success", 0).show();
                        SubActivity.this.startActivity(new Intent(SubActivity.this.getApplicationContext(), DriveActivity.class));
                        return;
                    }
                    Toast.makeText(SubActivity.this.getApplicationContext(), "OTP Auth Failed", 0).show();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    public static String getGoogleAuthenticatorBarcode(String secretKey, String issuer, String userName) throws Exception {
        return "otpauth://totp/" + URLEncoder.encode(issuer + ":" + userName, "UTF-8").replace("%", "%20") + "?secret=" + URLEncoder.encode(secretKey, "UTF-8").replace("%", "%20") + "&issuer=";
    }

    public static String getTOTPCode(String secretKey) {
        return TOTP.getOTP(Hex.encodeHexString(new Base32().decode(secretKey)));
    }
}

```

[그림 7] SubActivity 관련 코드

SubActivity의 getTOTPCode 함수의 Return 값을 equal 함수를 통해 OTPEditText 부분에 입력한 값을 비교하는 것을 확인할 수 있다. 이 조건에서 True가 나올 경우 OTP Auth Success라는 메시지가 출력된다.

이 경우 Simple하게 Equal 함수를 후킹하여 Equal 함수의 비교 대상 Parameter를 가져와서 OTP를 확인할 수 있다. 소스코드를 확인해본 결과 OTP 갱신이 30초 경과 시에만 이루어지기 때문에 30초 내로 Hooking이 진행된다면 가능하다. 해당 방법 구현은 [첨부 2]와 같이 F를 이용했다.

```

/* renamed from: de.taimos.totp.TOTP */
public final class TOTP {
    private TOTP() {
    }

    public static String getOTP(String key) {
        return getOTP(getStep(0, key));
    }

    public static boolean validate(String key, String otp) {
        return validate(getStep(), key, otp);
    }

    private static boolean validate(long step, String key, String otp) {
        return getOTP(step, key).equals(otp) || getOTP(step - 1, key).equals(otp);
    }

    private static long getStep() {
        return System.currentTimeMillis() / 30000;
    }

    private static String getOTP(long step, String key) {
        String steps = Long.toHexString(step).toUpperCase();
        while (steps.length() < 16) {
            steps = "0" + steps;
        }
        byte[] hash = hmac_sha1(hexStr2Bytes(key), hexStr2Bytes(steps));
        int offset = hash[hash.length - 1] & 15;
        String result = Integer.toString((((hash[offset + 1] & UByte.MAX_VALUE) << 16) | ((hash[offset] & ByteCompanionObject.MAX_VALUE) << 24)) | ((hash[offset + 2] & UByte.MAX_VALUE) << 8));
        while (result.length() < 6) {
            result = "0" + result;
        }
        return result;
    }
}

```

[그림 8] TOTP 관련 코드

```

import frida, sys, time

app_name = 'com.example.privatedrive'

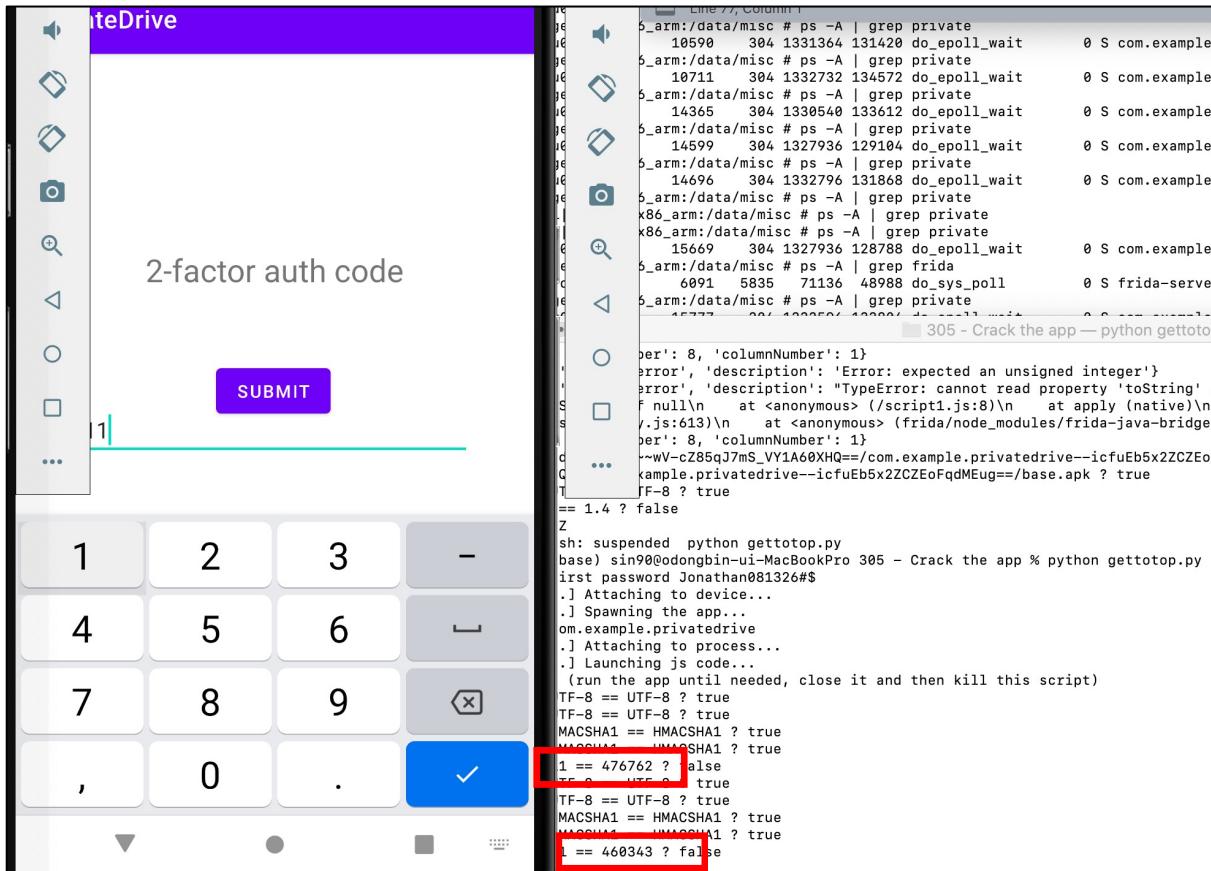
def on_message(message, data):
    print(message)

jscode = """
const str = Java.use('java.lang.String');
const objectClass = 'java.lang.Object';
str.equals.overload(objectClass).implementation = function(obj) {
    //Print the incoming parameters
    var response = str.equals.overload(objectClass).call(this, obj);
    var hasNumber = /\d/;
    var tset = hasNumber.test(obj.toString());
    if(tset) {
        console.log(str.toString.call(this) + ' == ' + obj.toString() + ' ? ' + response);
    }
    return response;
}
"""
print("[.] Attaching to device...")
try:
    device = frida.get_usb_device()
except:
    print("[-] Can't attach. Is the device connected?")
    sys.exit()
print("[.] Attaching to process...")
try:
    process = device.attach(10590) # ps로 pid를 구함
except:
    print("[-] Can't connect to App.")
    sys.exit()

print("[.] Launching js code...")
print(" (run the app until needed, close it and then kill this script)")
script = process.create_script(jscode)
script.on('message', on_message)
script.load()
try:
    sys.stdin.read()
except KeyboardInterrupt:
    print ("\nExiting now")
    exit(0)

```

[첨부 2] Frida 이용 해결 소스코드



[그림 9] Equal function Hooking

위와 같이 Hooking을 통해 비교 값을 통해 알아낼 수도 있지만 직접 Key derivation Algorithm을 분석하여 모방하는 Source Code를 작성하여 OTP를 생성할 수 있다. OTP를 생성과 관련된 부분은 아래와 같다.

```

/* renamed from: de.taimos.totp.TOTP */
public final class TOTP {
    ...
}

public static String getOTP(String key) {
    return getOTP(getStep(), key);
}

public static boolean validate(String key, String otp) {
    return validate(getStep(), key, otp);
}

private static boolean validate(long step, String key, String otp) {
    return getOTP(step, key).equals(otp) || getOTP(step - 1, key).equals(otp);
}

private static long getStep() {
    return System.currentTimeMillis() / 30000;
}

private static String getOTP(long step, String key) {
    String steps = Long.toHexString(step).toUpperCase();
    while (steps.length() < 16) {
        steps = "0" + steps;
    }
    byte[] hash = hmac_sha1(hexStr2Bytes(key), hexStrBytes(steps));
    int offset = hash[hash.length - 1] & 15;
    String result = Integer.toString((((hash[offset + 1] & UByte.MAX_VALUE) << 16) | ((hash[offset] & ByteCompanionObject.MAX_VALUE) << 24)) | ((hash[offset + 2] & UByte.MAX_VALUE) << 8));
    while (result.length() < 6) {
        result = "0" + result;
    }
    return result;
}

private static byte[] hexStr2Bytes(String hex) {
    byte[] bArray = new BigInteger("10" + hex, 16).toByteArray();
    byte[] ret = new byte[bArray.length - 1];
    System.arraycopy(bArray, 1, ret, 0, ret.length);
    return ret;
}

private static byte[] hmac_sha1(byte[] keyBytes, byte[] text) {
    try {
        Mac hmac = Mac.getInstance("HmacSHA1");
        hmac.init(new SecretKeySpec(keyBytes, "HmacSHA1"));
        return hmac.doFinal(text);
    } catch (GeneralSecurityException gse) {
        throw new UndeclaredThrowableException(gse);
    }
}

```

[그림 10] Key derivation Algorithm

위의 소스코드를 기반으로 분석한 Key derivation Algorithm 작동은 아래와 같다.

- 1. SecretKey : C0605R.string.password_key.substring(15).toUpperCase()
- 2. TOTP.getOTP Input 값 생성 : Hex.encodeHexString(Base32().decode(secretKey))
- 3. System.currentTimeMillis() / 30000 을 통해 HMAC SHA1 입력 값 생성
- 4. HMAC sha1의 해시값의 특정 오프셋 값들을 정해진 값들로 XOR 후 도출된 값을 OR 연산 및 1000000으로 나눈 나머지 산출
- 5. 글자수 6자리 미만일 경우 0으로 좌측 Padding

위의 로직은 First Password를 구한 것과 달리 비교 연산 대상을 구할 때 역연산이 필요가 없이 APK가 생성한 숫자 그대로를 재현하면 된다. 위의 로직을 이용하여 아래 JAVA 기반의 OTP 생성 Source code를 구현하였다. APK Decompile을 정상적으로 하였다면 해당 내용은 참고하여 쉽게 작성할 수 있다.

```
import java.lang.reflect.UndeclaredThrowableException;
import java.math.BigInteger;
import java.security.GeneralSecurityException;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import kotlin.UByte;
import kotlin.jvm.internal.ByteCompanionObject;
import org.apache.commons.codec.binary.Base32;
import org.apache.commons.codec.binary.Hex;

class TOTP {

    public static void main(String[] args) {
        String secretKey = "203d233e382c1e215a6a7c6c7e725c6b".substring(15).toUpperCase();
        System.out.println(secretKey);
        String result = TOTP.getOTP(Hex.encodeHexString(new Base32().decode(secretKey)));
        System.out.println(result);
    }

    public static String getOTP(String key) {
        System.out.println(key);
        return calculateOTP(getStep(), key);
    }

    private static long getStep() {
        long step = System.currentTimeMillis() / 30000;
        return step;
    }

    private static String calculateOTP(long step, String key) {
        String steps = Long.toHexString(step).toUpperCase();
        while (steps.length() < 16) {
            steps = "0" + steps;
        }
        byte[] hash = hmac_sha1(hexStr2Bytes(key), hexStr2Bytes(steps));
        int offset = hash[hash.length - 1] & 15;
        String result = Integer.toString((((hash[offset + 1] & UByte.MAX_VALUE) << 16) | ((hash[offset] & ByteCompanionObject.MAX_VALUE) << 24)) | ((hash[offset + 2] & UByte.MAX_VALUE) << 8)) | (hash[offset + 3] & UByte.MAX_VALUE) % 1000000;
        System.out.println(offset);
        while (result.length() < 6) {
            result = "0" + result;
        }
        return result;
    }

    private static byte[] hexStr2Bytes(String hex) {
        byte[] bArray = new BigInteger("10" + hex, 16).toByteArray();
        byte[] ret = new byte[(bArray.length - 1)];
        System.arraycopy(bArray, 1, ret, 0, ret.length);
        return ret;
    }

    private static byte[] hmac_sha1(byte[] keyBytes, byte[] text) {
        try {
            Mac hmac = Mac.getInstance("HmacSHA1");
            hmac.init(new SecretKeySpec(keyBytes, "RAW"));
            return hmac.doFinal(text);
        }
    }
}
```

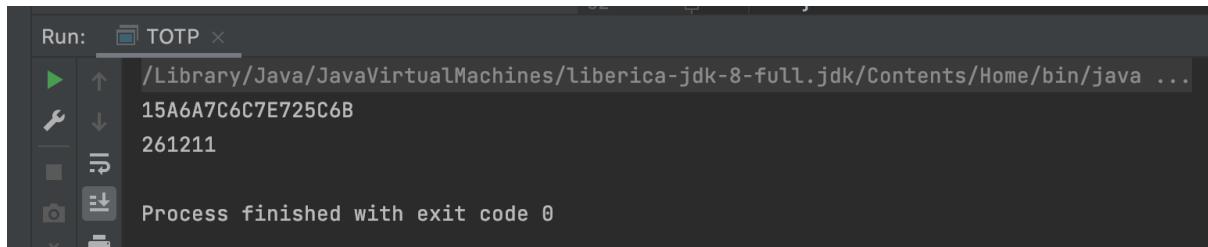
```

        } catch (GeneralSecurityException gse) {
            throw new UndeclaredThrowableException(gse);
        }
    }
}

```

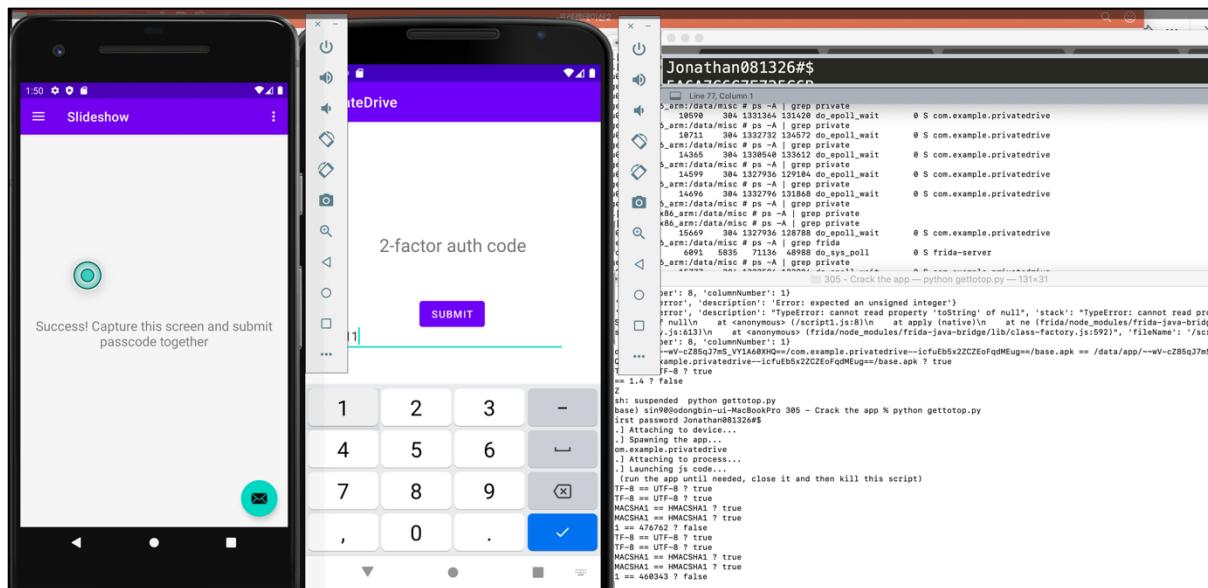
[첨부 3] Implementation of Key derivation Algorithm

위의 소스코드를 실행하면 아래의 그림과 같은 결과를 얻을 수 있고, 아래 문제 2와 같이 인증이 가능함을 확인할 수 있다.



[그림 11] OTP 획득

- After unlocking the passcode, take screenshots with the ‘first password’ and OTP code.



[그림 12] first password와 OTP 코드를 이용해 잠금 해제에 성공한 화면

획득한 Password (**Jonathan081326#\$**)를 입력하여 로그인에 성공한 뒤, 2-factor auth code 입력 화면으로 넘어온 화면이다. 또한 VM 2대를 이용하여 중간 VM에서 나온 OTP 코드 (460343)를 좌측 VM에 입력하여 원하는 Screenshot을 얻을 수 있었다.