1. Unpack the apk file.

```
┌──(kali㉿kali)-[~/Desktop/geologin]
└─$ ls
geologin.apk

┌──(kali㉿kali)-[~/Desktop/geologin]
└─$ cp geologin.apk geologin.zip

┌──(kali㉿kali)-[~/Desktop/geologin]
└─$ mkdir geologin-unzip

┌──(kali㉿kali)-[~/Desktop/geologin]
└─$ unzip geologin.zip -d ~/Desktop/geologin/geologin-unzip
Archive:  geologin.zip
  inflating: /home/kali/Desktop/geologin/geologin-unzip/Andro
```

2. Convert “classes.dex” to a jar file with dex2jar (https://github.com/pxb1988/dex2jar)

```
┌──(kali㉿kali)-[~/Desktop/geologin/geologin-unzip]
└─$ ls
AndroidManifest.xml  classes.dex  META-INF  res  resources.arsc

┌──(kali㉿kali)-[~/Desktop/geologin/geologin-unzip]
└─$ d2j-dex2jar classes.dex
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
dex2jar classes.dex → ./classes-dex2jar.jar

┌──(kali㉿kali)-[~/Desktop/geologin/geologin-unzip]
└─$
```

3. Analyze the “LoginViewModel” Class in JD-GUI
   The following is noticeable:
   a. A String is created with two “textView” values & a parameter (the entered PIN). The integers of the “textView” values can be searched for in “R.class”. They represent the Country & Region. The string should be “CH.Zermatt.<PIN>”.
   The Country (CH) & Region (Zermatt) was derived from the challenge description.

You must login into the app to get the flag. But be aware, the login is restricted to people located in Zermatt (CH) only. As the app trust all people in Zermatt, a 4 digit PIN will do.

```
ublic void login(Activity paramActivity, String paramString) {
  if (!hasInternet(paramActivity)) {
    this.loginResult.setValue(new LoginResult(Integer.valueOf(2131624027)));
    return;
  }
  if (this.ipCountryLocation != null) {
    try {
      TextView textView2 = (TextView)paramActivity.findViewById(2131230833);
      TextView textView1 = (TextView)paramActivity.findViewById(2131230982);
      String str = String.format("%s.%s.%s", new Object[] { textView2.getText(), textView1.getText(), paramString
      LoginTask loginTask = new LoginTask();
      this(this, null);
      loginTask.execute(new String[] { String.format("%s/%s", new Object[] { checkURL, getMD5(str) }) });
    } catch (Exception paramActivity) {
      this.loginResult.setValue(new LoginResult(Integer.valueOf(2131623978)));
    }
  } else {
    this.loginResult.setValue(new LoginResult(Integer.valueOf(2131624037)));
  }
```

b. A LoginTask is executed with the concatenated parameters of: *the "checkURL" variable, a forward slash, and the MD5 hash of the string*. The "checkURL" variable is declared in the beginning so with that string prefix, the concatenated string should be "https://ja3er.com/img/<MD5 HASH OF PREVIOUS STRING>".

```
public class LoginViewModel extends ViewModel {
  private static final char[] HEX_ARRAY = "0123456789ABCDEF".toCharArray();

  public static String checkURL = "https://ja3er.com/img";

  private String ipCheckUrl = "https://ipinfo.io/json";

  private String ipCountryLocation = null;

  private String ipRegionLocation = null;
```

c. If we analyze byte code ("doInBackground") in the previously called "LoginTask" class, we can notice some words/strings that point to a web request being made & the output stream being converted to a string.

```
private class LoginTask extends AsyncTask<String, Void, String> {
  private LoginTask() {}

  private String getStringFromStream(InputStream paramlInputStream) {
    BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(paramlInputStream));
    StringBuilder stringBuilder = new StringBuilder();
    try {
      while (true) {
        String str = bufferedReader.readLine();
        if (str != null) {
          stringBuilder.append(str);
          stringBuilder.append('\n');
          continue;
        }
        break;
      }
      try {
        paramlInputStream.close();
      } catch (IOException paramlInputStream) {
        paramlInputStream.printStackTrace();
      }
    } catch (IOException iOException) {
      iOException.printStackTrace();
      paramlInputStream.close();
    } finally {}
    return stringBuilder.toString();
  }

  protected String doInBackground(String... paramlVarArgs) { // Byte code:
    //   0: new java/net/URL
    //   3: astore_2
    //   4: aload_2
    //   5: aload_1
    //   6: iconst_0
    //   7: aaload
    //   8: invokespecial <init> : (Ljava/lang/String;)V
    //   11: aload_2
    //   12: invokevirtual openConnection : ()Ljava/net/URLConnection;
    //   15: checkcast java/net/HttpURLConnection
    //   18: astore_2
    //   19: aload_2
    //   20: astore_1
    //   21: aload_2
    //   22: invokevirtual connect : ()V
    //   25: aload_2
    //   26: astore_1
    //   27: aload_0
    //   28: aload_2
    //   29: invokevirtual getInputStream : ()Ljava/io/InputStream;
    //   32: invokespecial getStringFromStream : (Ljava/io/InputStream;)Ljava/lang/String;
    //   35: astore_3
    //   36: aload_2
    //   37: ifnull -> 44
    //   40: aload_2
    //   41: invokevirtual disconnect : ()V
```

4. From the information gathered we can assume that a String (“CH.Zermatt.<ENTERED PIN>”) is created. This string is then hashed with MD5. The MD5 Hash is appended to the URL “https://ja3er.com/img/”. The full URL (“https://ja3er.com/img/<MD5 HASH>”) is requested & the output is converted to a string (supposedly the flag).
   Based on this information we can write a python script (“pin_bruteforce.py”) to hash every PIN with the string prefix (“CH.Zermatt.”) & send a request to the URL (with the MD5 Hash appended).

pin_bruteforce.py > ...

```python
import requests
import hashlib

# Sample URL: https://ja3er.com/img/e392078756e65caa2475ca70273658a9
# MD5 hashed String: CH..1111 (no "Zermatt" due to old code,
#                              which only allowed null region (/no region)
#                              to login.)
#
# Bruteforce Pattern: CH.Zermatt.<4-Digit-Pin>
#
# This script bruteforces all possible combinations of
# a 4 digit pin with the "CH.Zermatt." prefix (appended to the URL prefix).
# The prefix can be derived from the java application code.


def domd5hash(string):
    md5_hash = hashlib.md5(string.encode())
    return md5_hash.hexdigest()


def getUrl(md5_str):
    check_url = "https://ja3er.com/img/"
    url = check_url + md5_str
    return url


def doReq(url):
    status = requests.get(url).status_code
    if status == 404:
        return "wrong"
    else:
        return "correct"


def main():
    for i in range(10000):
        pin = "{0:04d}".format(i)
        print("Trying pin: ", pin)
        md5_inp = "CH.Zermatt." + str(pin)
        md5 = domd5hash(md5_inp)
        url = getUrl(md5)
        req_try = doReq(url)
        if req_try == "correct":
            print("Correct pin: " + pin)
            print("Correct URL:" + url)
            print("Output (Flag): " + requests.get(url).text)
            break
        else:
            continue


if __name__ == "__main__":
    main()

# Check if MD5 hash is identical to Sample URL Hash:
# print(domd5hash("CH..1111"))
```

5. The Output is the correct PIN & URL, as well as the flag (response output).

```
Trying pin:  2017
Trying pin:  2018
Trying pin:  2019
Trying pin:  2020
Trying pin:  2021
Correct pin: 2021
Correct URL:https://ja3er.com/img/d3cfe142d34095520f6b776e864278ab
Output (Flag): SCS21{R3wire-Your-Br@in-Cr@ck-The-Ch@lleng3s}
```