**Signing Android Applications**

***Applications to be tested on the Emulator must be signed using the secure cabinet: “debug.keystore”.***

***It should contain a key named: “androiddebugkey”.***

**NOTE**

The public/private Android key container –called *debug.keystore*- is saved in the following folder (assuming r1.5)

Windows XP:

C:\Documents and Settings\Administrator\.android\debug.keystore

Default phone memory image and SD card:

C:\Documents and Settings\Administrator\.android\avd\AvdApi3Id3MatosSDcard.avd\userdata-qwmu.img

C:\Documents and Settings\Administrator\.android\avd\AvdApi3Id3MatosSDcard.avd\userdat.img (???)

Windows Vista: C:\Users\Administrator\.android\

Mac OS /Userstheuser/.android/debug.keystore

Eclipse IDE points to the default debug.keystore file. If you want to check it or change it to a custom keystore, follow the sequence : **Window | Preferences | Android | Build**. Under “Default debug keystore” you will find the reference:

**C:\Documents and Settings\Administrator\.android\debug.keystore**

If needed click on *Browse* and move to the appropriated folder holding the custom keystore.

Mac OS users: Should follow the sequence **Eclipse| Preferences | Android | Build.**

***CAUTION:****THIS MATERIAL WORKS WELL WHEN EXECUTING FROM THE JAVA SDK* ***/BIN*** *SUBDIRECTORY. WHEN EXECUTING FROM OTHER PLACES IT MAY GIVE SILLY ERRORS (MOST LIKELY RELATED TO PROBLEMS WITH DOS PATH, JAVAPATH AND ALL THAT NON-SENSE).*

**Signing / Publishing Android Applications.**

Before anything is signed you need to provide *naming/versioning* information into the application code. To do that you should define two special entries (*versionCode,* vesionName) in the Manifest file. The following example shows the use of android:versionCode and android:versionName attributes.

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="com.example.package.name"

**android:versionCode="2"**

**android:versionName="1.1"**>

<application android:icon="@drawable/icon" android:label="@string/app\_name">

...

</application>

</manifest>

**STEP1. Create a Keystore**

*Keystores* are used to support a secure computing environment. A ***keystore*** file is a cryptographic database file that contains matching sets of both *public* keys and *private* keys. The container, as well as each set of key values is password protected.  The private key is kept secret, while the public key can be given to anybody. When two parties want to securely communicate the sender must encrypt the message using the well known recipient’s public key.

Incoming messages ([encrypted](http://en.wikipedia.org/wiki/Encryption) with the recipient's public key) can only be decrypted with the corresponding private key (which is known by the receiving side). The keys are related in a non-trivial mathematical way, and the private key cannot be practically derived from the public key.

To create a *keystore* you should use the ***keytool*** program which is part of the Java SDK. Make sure your PATH environment variable allows access to the most current Java-SDK.

**Example**:

Suppose we want to sign an application called: *IAMOK1.APK*. Assume *naming/versioning* has already been taken care of by inserting proper attributes in the application’s manifest. To be able to sign this app (and other future apps) we need to first obtain a set of <private/public> keys.

In this example we will create a vault or keystore called “***debug.keystore****”* (this is the default name expected by the Eclipse environment. To see or change the (Eclipse) path do: Windows > Preferences > Android (expand+) > Build. Two boxes point to the default vault and optional custom made keystore)*.*

The key to open the *debug.keystore* container is “android”. Inside we will create and store a public key called “androiddebugkey”. For simplicity we choose the password to the public portion for this key entry to be “android”. The corresponding private key will be created using the RSA algorithm and we will request 10000 days of longevity. The command call should be entered in a single line; in our example we break into several lines to enhance readability.

**keytool -genkey –v**

**-keystore debug.keystore**

**-alias androiddebugkey**

**-keyalg RSA**

**-validity 10000**

**-storepass android**

**-keypass android**

During this process you will be prompted to enter personal information such as name, affiliation, location, etc. This will appear in connection to the secure self-signed certificate to be created. In our example we see the following dialog

What is your first and last name?

[Unknown]: victor matos

What is the name of your organizational unit?

[Unknown]: cis department

What is the name of your organization?

[Unknown]: cleveland state university

What is the name of your City or Locality?

[Unknown]: cleveland

What is the name of your State or Province?

[Unknown]: ohio

What is the two-letter country code for this unit?

[Unknown]: us

Is CN=victor matos, OU=cis department, O=cleveland state university, L=cleveland, ST=ohio, C=us correct?

[no]: yes

Generating 1,024 bit RSA key pair and self-signed certificate (SHA1withRSA) with a validity of 10,000 days

for: CN=victor matos, OU=cis department, O=cleveland state university, L=cleveland, ST=ohio, C=us

[Storing debug.keystore]

At the end of this process a file called *debug.keystore* should have been created in you file system.



**STEP2. Verify the keystore – See your MD5 fingerprint**

For peace-of-mind you should verify the quality of your keystore. This process is also used in displaying the vault’s *fingerprint.* The certificate’s fingerprint value will be used in other validation procedures such as obtaining a Google’s *android:apiKey* (required for an Android application that wants to get access to the Google-Map Service).

In our example we enter the following command (one line)

**keytool -list -keystore debug.keystore**

in this example it returns:

Enter keystore password:

Keystore type: JKS

Keystore provider: SUN

Your keystore contains 1 entry

androiddebugkey, Jan 22, 2009, PrivateKeyEntry,

Certificate fingerprint (MD5): 7E:E2:C2:12:FB:58:8E:EF:2A:BF:EB:85:49:6A:84:52

This output indicates that the keystore has *one* entry in the <public,private> collection of keys. In this case the public key is **androiddebugkey** and it has its corresponding matching (non-displayable) *PrivateKeyEntry* for which a certificate has been made.

**Certificates**

A public key certificate (also known as a *digital certificate* or *identity certificate*) is an electronic document which uses a [digital signature](http://en.wikipedia.org/wiki/Digital_signature) (fingerprint) to bind together a [public key](http://en.wikipedia.org/wiki/Public_key) with an identity — information such as the name of a person or an organization, their address, and so forth. The certificate can be used to verify that a public key belongs to an individual.

The output above shows the fingerprint associated to the public key. You may optionally create a *Certificate File* which summarizes information about a single public key. This is useful in SSL exchange (such as when using Google Maps API). If you choose to create the certificate enter the following command:

**keytool -exportcert -v**

**-alias androiddebugkey**

**-keystore debug.keystore > MyAndroidDebugCertificate2019.crt**

Enter keystore password: android

Inspect contents of the optional certificate file using the following command

**keytool -printcert -v -file myAndroidDebugCertificate2019.crt**

You should see output like this

Owner: CN=Victor Matos, OU=CIS Dept, O=CSU, L=Cleveland, ST=Ohio, C=US

Issuer: CN=Victor Matos, OU=CIS Dept, O=CSU, L=Cleveland, ST=Ohio, C=US

Serial number: 4af9b5cf

Valid from: Tue Nov 10 13:49:51 EST 2009 until: Fri Nov 08 13:49:51 EST 2019

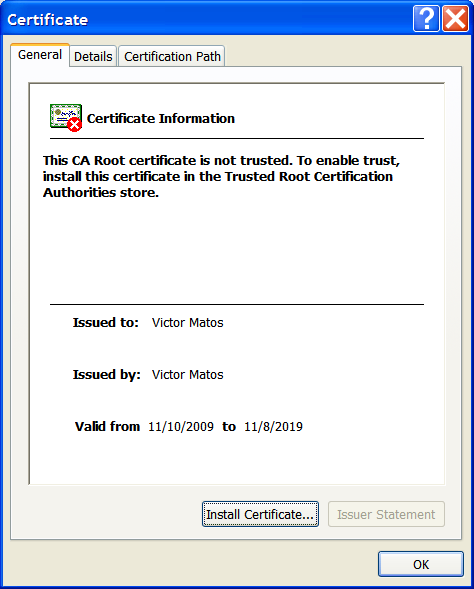
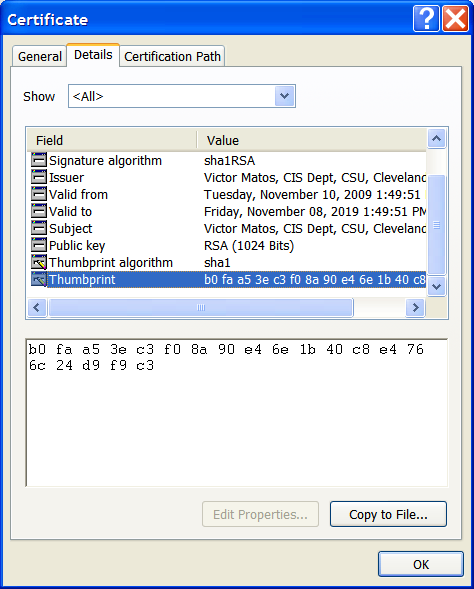
Certificate fingerprints:

MD5: 71:73:97:14:13:13:FB:E5:38:A9:B3:9C:61:9C:56:F1

SHA1: B0:FA:A5:3E:C3:F0:8A:90:E4:6E:1B:40:C8:E4:76:6C:24:D9:F9:C3

Signature algorithm name: SHA1withRSA

Version: 3





**STEP3. Signing Your Android Apk**

In order to distribute your application it must be first processed by the ***jarsigner*** program which is part of the Java SDK.

*The****jarsigner****tool is used in Android for two purposes: (1) to sign Java bytecode (.APK) files, and (2) to verify the signatures and integrity of signed .APK files. Archived APK files contain class files, images, sounds, and other digital data in a single file for faster and easier distribution. When processed by jarsigner an .APK files will also contain a META-INF/MANIFEST.MF file.*

*A****digital signature****is a string of bits that is computed from some data (the APK being "signed") and the private key of an entity. In order for an entity's signature to be generated for a file, the entity must first have a public/private key pair associated with it, and also one or more certificates authenticating its public key.*

*A* ***certificate****is a digitally signed statement from one entity, saying that the public key of some other entity has a particular value.*

*jarsigner uses key and certificate information from a****keystore****to generate digital signatures for APK files.*

*jarsigner uses an entity's private key to generate a signature. The signed APK file contains, among other things, a copy of the certificate from the keystore for the public key corresponding to the private key used to sign the file.*

*jarsigner can verify the digital signature of the signed JAR file using the certificate inside it (in its signature block file).*

In our example we use the following command (it must be a single-line command). It asks jarsigner to open the keystore known as “**debug.keystore**” using the keypass “**android**”. Inside the vault it should look for the public key “**androiddebugkey**” and obtain its corresponding private value using the keypass “**android**”. Finally, all this information will be use to sign the Android application known as “**IamOk1.apk**” (the following command should be written on only one line).

**jarsigner -verbose**

**-keystore debug.keystore**

**-storepass android**

**-keypass android**

**IamOk1.apk**

**androiddebugkey**

After executing the command you will see something like the lines below indicating the aplication’s apk now contains a secure digital signature agreeable to the Android emulator (or device)

adding: META-INF/VMATOSKE.SF

adding: META-INF/VMATOSKE.RSA

signing: res/drawable/btn\_dropdown\_down.9.png

signing: res/drawable/icon.png

signing: res/drawable/picture\_emergency.png

signing: res/layout/main.xml

signing: res/layout/mycontacts\_layout.xml

signing: AndroidManifest.xml

signing: resources.arsc

signing: classes.dex

**STEP4. Verifying Signature on the Android Application**

**jarsigner -verify -verbose -certs IamOk1.apk**

You will see something like:

401 Mon Oct 19 16:16:30 EDT 2009 META-INF/MANIFEST.MF

541 Thu Nov 12 09:48:10 EST 2009 META-INF/ANDROIDD.SF

925 Thu Nov 12 09:48:10 EST 2009 META-INF/ANDROIDD.RSA

454 Mon Oct 19 16:16:30 EDT 2009 META-INF/CERT.SF

776 Mon Oct 19 16:16:30 EDT 2009 META-INF/CERT.RSA

sm 3366 Mon Oct 19 15:11:08 EDT 2009 res/drawable/icon.png

X.509, CN=Victor Matos, OU=CIS Dept, O=CSU, L=Cleveland, ST=Ohio, C=US

[certificate is valid from 11/10/09 1:49 PM to 11/8/19 1:49 PM]

X.509, CN=Android Debug, O=Android, C=US

[certificate expired on 11/4/09 10:23 PM]

sm 916 Mon Oct 19 16:16:30 EDT 2009 res/layout/main.xml

X.509, CN=Victor Matos, OU=CIS Dept, O=CSU, L=Cleveland, ST=Ohio, C=US

[certificate is valid from 11/10/09 1:49 PM to 11/8/19 1:49 PM]

X.509, CN=Android Debug, O=Android, C=US

[certificate expired on 11/4/09 10:23 PM]

sm 1416 Mon Oct 19 16:16:30 EDT 2009 AndroidManifest.xml

X.509, CN=Victor Matos, OU=CIS Dept, O=CSU, L=Cleveland, ST=Ohio, C=US

[certificate is valid from 11/10/09 1:49 PM to 11/8/19 1:49 PM]

X.509, CN=Android Debug, O=Android, C=US

[certificate expired on 11/4/09 10:23 PM]

sm 1156 Mon Oct 19 15:11:08 EDT 2009 resources.arsc

X.509, CN=Victor Matos, OU=CIS Dept, O=CSU, L=Cleveland, ST=Ohio, C=US

[certificate is valid from 11/10/09 1:49 PM to 11/8/19 1:49 PM]

X.509, CN=Android Debug, O=Android, C=US

[certificate expired on 11/4/09 10:23 PM]

sm 4388 Mon Oct 19 16:16:30 EDT 2009 classes.dex

X.509, CN=Victor Matos, OU=CIS Dept, O=CSU, L=Cleveland, ST=Ohio, C=US

[certificate is valid from 11/10/09 1:49 PM to 11/8/19 1:49 PM]

X.509, CN=Android Debug, O=Android, C=US

[certificate expired on 11/4/09 10:23 PM]

s = signature was verified

m = entry is listed in manifest

k = at least one certificate was found in keystore

i = at least one certificate was found in identity scope

jar verified.

**STEP5. Install your signed APK application in the Android Emulator (or device).**

Use the Android-Debug-Bridge program available in the Android-SDK.

**adb** install **IamOk1.apk**

You will see something like:

763 KB/s (0 bytes in 24441.000s)

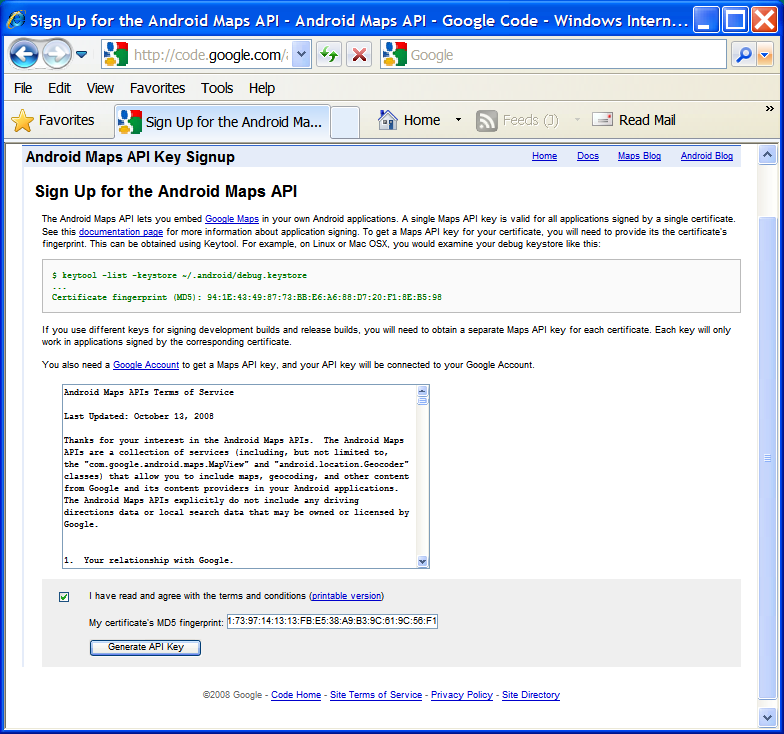
pkg: /data/local/tmp/IamOk1.apk

Success

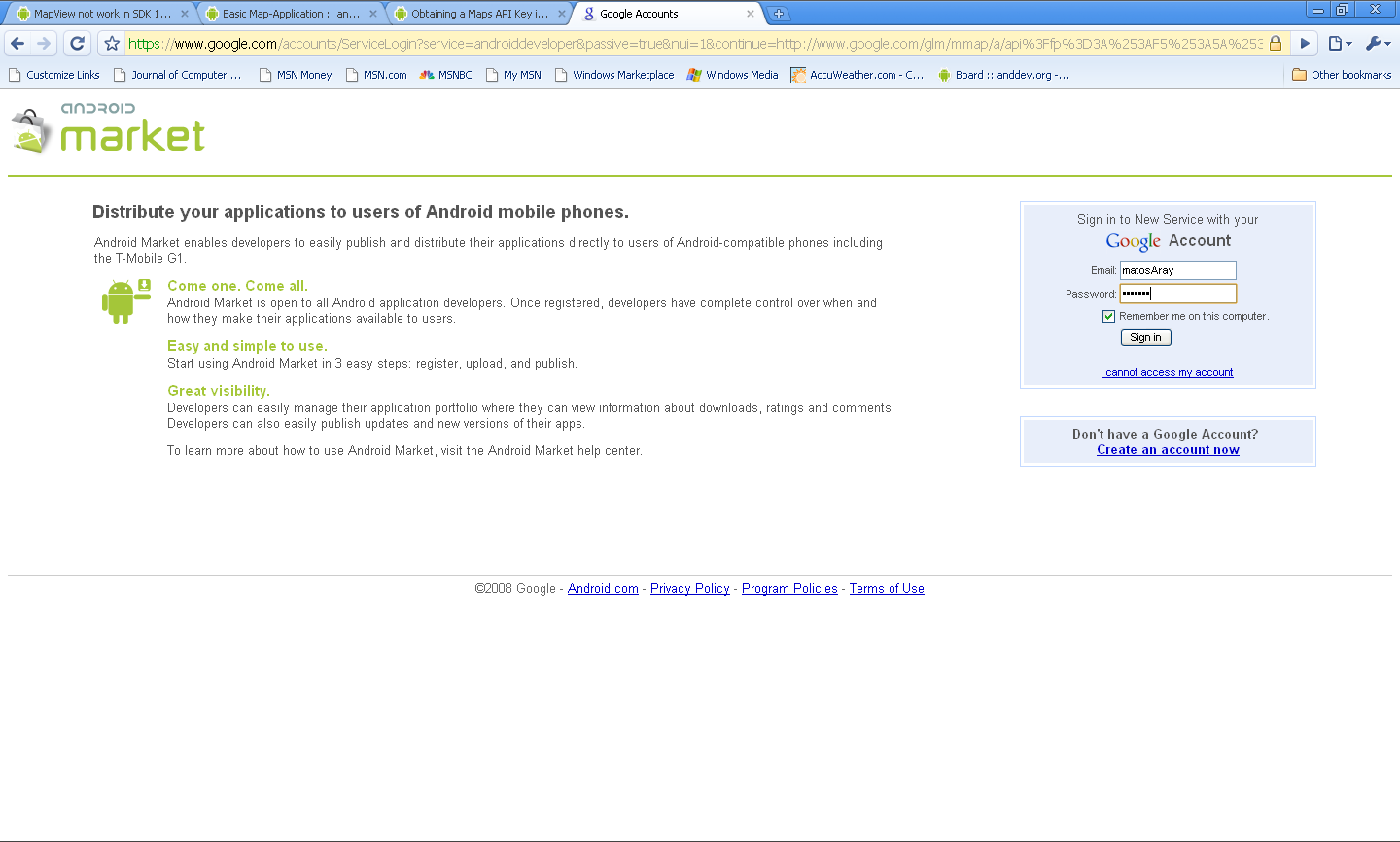
At this point you should be ready to try the application.

### Step6. Android Maps API Key Signup

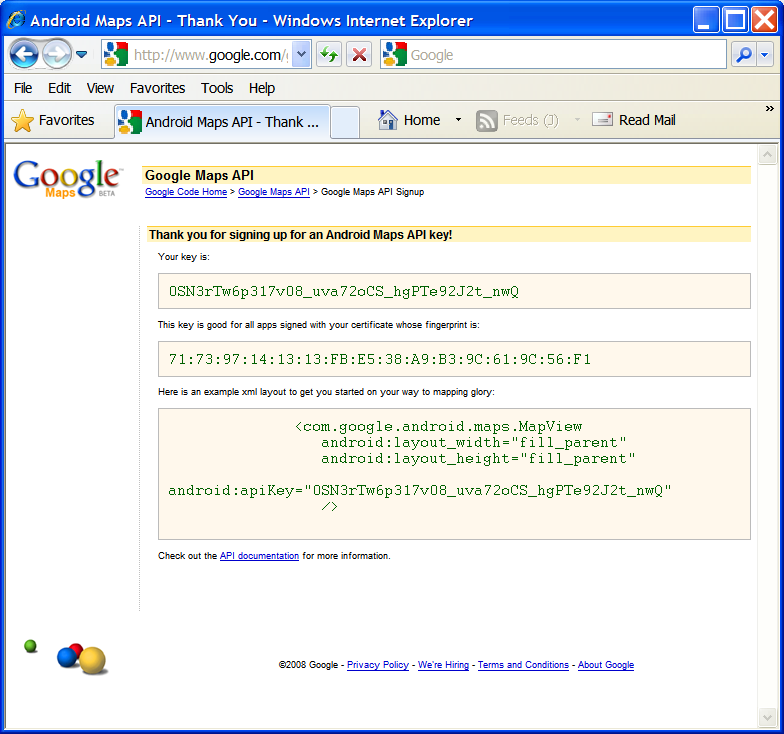
1. We assume you are a registered *Google developer* or at least have a *gmail* account. Therefore you know your *username/password* credentials. If you do not have them you will have an opportunity to do so in the next steps.
2. Now you go into the “**Sign Up for the Android Maps API”** process. Contact Google at <http://code.google.com/android/maps-api-signup.html>. Accept the service agreement and paste the MD5 fingerprint value obtained earlier. Click on *Generate API key* button:



3. The first time you try this, you will be prompted to enter your Google/Gmail login credentials (*username/password*).



4. Assuming everything went well you will see a screen like the following.



In this example the apikey value is

**0SN3rTw6p317v08\_uva72oCS\_hgPTe92J2t\_nwQ**

(this is valid for any applications generated under the fingerprint 71:73:97:14:13:13:FB:E5:38:A9:B3:9C:61:9C:56:F1)

When using a **MapView** attach the following signed fragment of xml to the layout/main.xml

<com.google.android.maps.MapView

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:apiKey="0SN3rTw6p317v08\_uva72oCS\_hgPTe92J2t\_nwQ"

/>

**Personal Notes for Room BU342**

**SDK1.5** uses the folder: **C:\Documents and Settings\Administrator\.android** for hosting *certificate* and *debug.keystore*.

The instructions to create the *fingerprint* and *ApiKey* are similar to the above for SDK1.0 except for the use of the sub-directory.

**MAPS - UPGRADING**

if you already have an ApiKey obtained for SDK1.0 then the (old) *keystore* is saved in the folder:

**C:\Documents and Settings\Administrator\Local Settings\Application Data\Android**

**You do not need to obtain a new key !!!**

Copy all the OLD files into the NEW destination in which SDK1.5 expects to see the keystore, the new destination is

C:\Documents and Settings\Administrator\.android

In my case:

**LAPTOP android:apiKey="0SN3rTw6p317v08\_uva72oCS\_hgPTe92J2t\_nwQ"**

**Note: SDK1.5**

If you do not have a previous Google-maps apiKey then follow the instructions from the beginning of these notes, remember that under SDK1.5 the *keystore* file is saved in the folder: C:\Documents and Settings\Administrator\.android

**Appendix A. Visual Tool to Create and Print a Certificate**.

*The following steps are optional. You could create the debug.keystore, obtain a fingerprint, sign Android applications, and request a Google* ***android:apikey*** *using the instructions given above.*

However, you may also use **KeyTool** **IUI** to generate and print the needed fingerprint.

* A graphical version of KeyTool software is available at: <http://yellowcat1.free.fr/index_ktl.html>.
* Download *ktl214.rar* (or latest Windows version), unpack the file, and execute the batch file: *run\_ktl.bat*.

The process to create the fingerprint is as follows:

1. Go to folder

**C:\Documents and Settings\Administrator\.android**

2. Make a copy of the file: debug.keystore and save it as

***debug.keystore.jks***

3. Run KeyTool IUI

Export | Private Keys's first certificate in chain | As Simple certificate file

4. On the screen enter

SOURCE

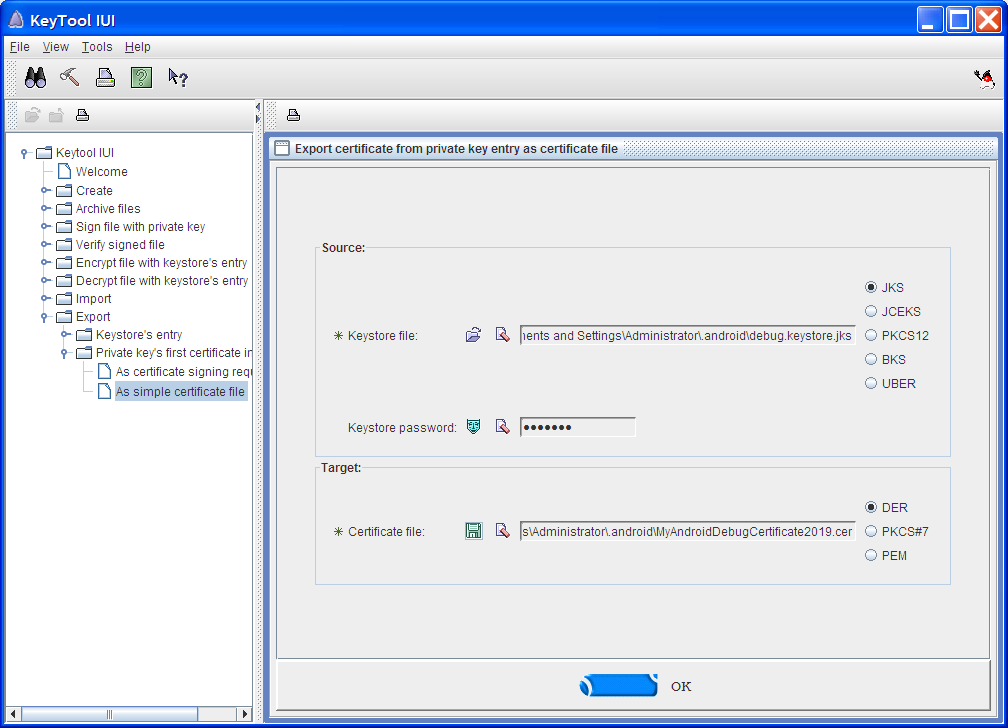
Keystore file: C:\Documents and Settings\Administrator\Local Settings\Application Data\Android\

***debug.keystore.jks***

keystore passwd: ***android***

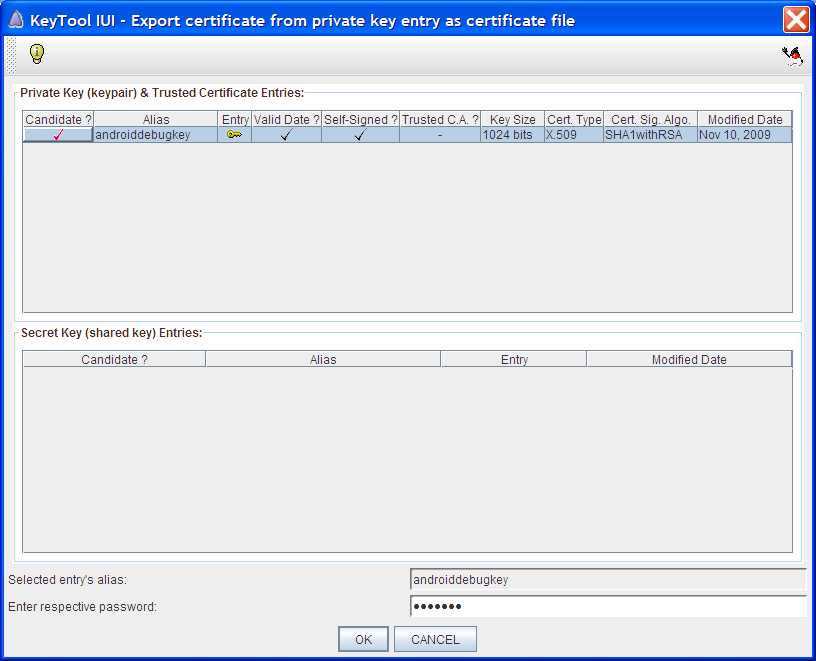
TARGET

Certificate file: ***MyAndroidDebugCertificate2019***

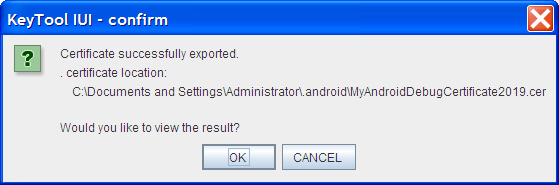


5. Click **OK** button

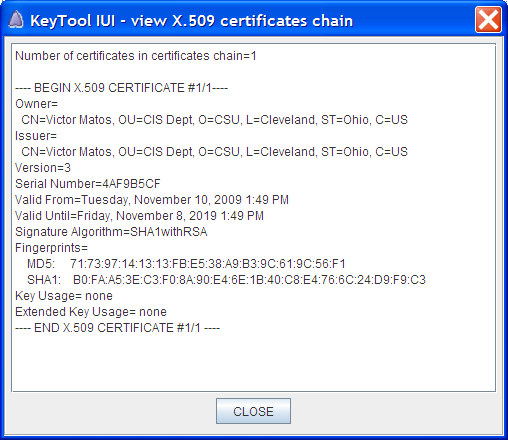
6. Select the first row: *androiddebugkey*. Enter respective password: ***android.*** Click OK button.



7. You will see a messagebox. Click **OK** to see the certificate file.



8. See the resulting values. For your future convenience copy to a text file (c:\myfingerprint.txt) the **MD5 Fingerprint** value from the .cer file



Number of certificates in certificates chain=1

---- BEGIN X.509 CERTIFICATE #1/1----

Owner=

CN=Victor Matos, OU=CIS Dept, O=CSU, L=Cleveland, ST=Ohio, C=US

Issuer=

CN=Victor Matos, OU=CIS Dept, O=CSU, L=Cleveland, ST=Ohio, C=US

Version=3

Serial Number=4AF9B5CF

Valid From=Tuesday, November 10, 2009 1:49 PM

Valid Until=Friday, November 8, 2019 1:49 PM

Signature Algorithm=SHA1withRSA

Fingerprints=

MD5: 71:73:97:14:13:13:FB:E5:38:A9:B3:9C:61:9C:56:F1

SHA1: B0:FA:A5:3E:C3:F0:8A:90:E4:6E:1B:40:C8:E4:76:6C:24:D9:F9:C3

Key Usage= none

Extended Key Usage= none

---- END X.509 CERTIFICATE #1/1 ----

10. In this example the value MD5 Fingerprint value is= 71:73:97:14:13:13:FB:E5:38:A9:B3:9C:61:9C:56:F1

**Appendix B. Keytool Command**

keytool usage:

-certreq [-v] [-protected]

[-alias <alias>] [-sigalg <sigalg>]

[-file <csr\_file>] [-keypass <keypass>]

[-keystore <keystore>] [-storepass <storepass>]

[-storetype <storetype>] [-providername <name>]

[-providerclass <provider\_class\_name> [-providerarg <arg>]] ...

[-providerpath <pathlist>]

-changealias [-v] [-protected] -alias <alias> -destalias <destalias>

[-keypass <keypass>]

[-keystore <keystore>] [-storepass <storepass>]

[-storetype <storetype>] [-providername <name>]

[-providerclass <provider\_class\_name> [-providerarg <arg>]] ...

[-providerpath <pathlist>]

-delete [-v] [-protected] -alias <alias>

[-keystore <keystore>] [-storepass <storepass>]

[-storetype <storetype>] [-providername <name>]

[-providerclass <provider\_class\_name> [-providerarg <arg>]] ...

[-providerpath <pathlist>]

-exportcert [-v] [-rfc] [-protected]

[-alias <alias>] [-file <cert\_file>]

[-keystore <keystore>] [-storepass <storepass>]

[-storetype <storetype>] [-providername <name>]

[-providerclass <provider\_class\_name> [-providerarg <arg>]] ...

[-providerpath <pathlist>]

-genkeypair [-v] [-protected]

[-alias <alias>]

[-keyalg <keyalg>] [-keysize <keysize>]

[-sigalg <sigalg>] [-dname <dname>]

[-validity <valDays>] [-keypass <keypass>]

[-keystore <keystore>] [-storepass <storepass>]

[-storetype <storetype>] [-providername <name>]

[-providerclass <provider\_class\_name> [-providerarg <arg>]] ...

[-providerpath <pathlist>]

-genseckey [-v] [-protected]

[-alias <alias>] [-keypass <keypass>]

[-keyalg <keyalg>] [-keysize <keysize>]

[-keystore <keystore>] [-storepass <storepass>]

[-storetype <storetype>] [-providername <name>]

[-providerclass <provider\_class\_name> [-providerarg <arg>]] ...

[-providerpath <pathlist>]

-help

-importcert [-v] [-noprompt] [-trustcacerts] [-protected]

[-alias <alias>]

[-file <cert\_file>] [-keypass <keypass>]

[-keystore <keystore>] [-storepass <storepass>]

[-storetype <storetype>] [-providername <name>]

[-providerclass <provider\_class\_name> [-providerarg <arg>]] ...

[-providerpath <pathlist>]

-importkeystore [-v]

[-srckeystore <srckeystore>] [-destkeystore <destkeystore>]

[-srcstoretype <srcstoretype>] [-deststoretype <deststoretype>]

[-srcstorepass <srcstorepass>] [-deststorepass <deststorepass>]

[-srcprotected] [-destprotected]

[-srcprovidername <srcprovidername>]

[-destprovidername <destprovidername>]

[-srcalias <srcalias> [-destalias <destalias>]

[-srckeypass <srckeypass>] [-destkeypass <destkeypass>]]

[-noprompt]

[-providerclass <provider\_class\_name> [-providerarg <arg>]] ...

[-providerpath <pathlist>]

-keypasswd [-v] [-alias <alias>]

[-keypass <old\_keypass>] [-new <new\_keypass>]

[-keystore <keystore>] [-storepass <storepass>]

[-storetype <storetype>] [-providername <name>]

[-providerclass <provider\_class\_name> [-providerarg <arg>]] ...

[-providerpath <pathlist>]

-list [-v | -rfc] [-protected]

[-alias <alias>]

[-keystore <keystore>] [-storepass <storepass>]

[-storetype <storetype>] [-providername <name>]

[-providerclass <provider\_class\_name> [-providerarg <arg>]] ...

[-providerpath <pathlist>]

-printcert [-v] [-file <cert\_file>]

-storepasswd [-v] [-new <new\_storepass>]

[-keystore <keystore>] [-storepass <storepass>]

[-storetype <storetype>] [-providername <name>]

[-providerclass <provider\_class\_name> [-providerarg <arg>]] ...

[-providerpath <pathlist>]

**Appendix C. Jarsigner Command**

Usage: jarsigner [options] jar-file alias

jarsigner -verify [options] jar-file

[-keystore <url>] keystore location

[-storepass <password>] password for keystore integrity

[-storetype <type>] keystore type

[-keypass <password>] password for private key (if different)

[-sigfile <file>] name of .SF/.DSA file

[-signedjar <file>] name of signed JAR file

[-digestalg <algorithm>] name of digest algorithm

[-sigalg <algorithm>] name of signature algorithm

[-verify] verify a signed JAR file

[-verbose] verbose output when signing/verifying

[-certs] display certificates when verbose and verifying

[-tsa <url>] location of the Timestamping Authority

[-tsacert <alias>] public key certificate for Timestamping Authority

[-altsigner <class>] class name of an alternative signing mechanism

[-altsignerpath <pathlist>] location of an alternative signing mechanism

[-internalsf] include the .SF file inside the signature block

[-sectionsonly] don't compute hash of entire manifest

[-protected] keystore has protected authentication path

[-providerName <name>] provider name

[-providerClass <class> name of cryptographic service provider's

[-providerArg <arg>]] ... master class file and constructor argument