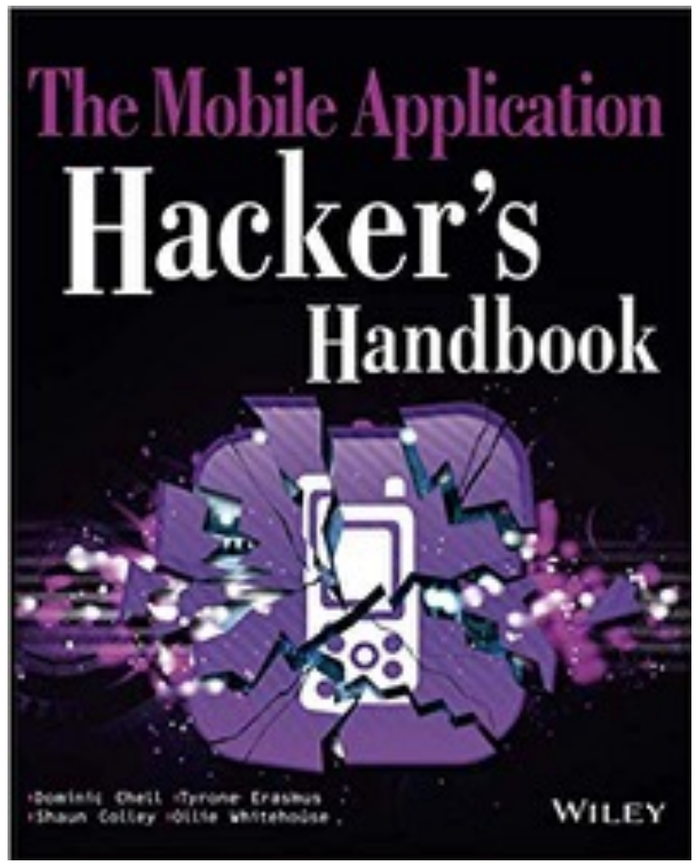
CNIT 128
Hacking Mobile Devices



6. Analyzing Android Applications Part 2

Updated 9-12-22

Topics

- Part 1
  - Creating Your First Android Environment
  - Understanding Android Applications
- Part 2
  - Understanding the Security Model: p 205-222
- Part 3
  - Understanding the Security Model: p 222ff
  - Reverse-Engineering Applications

Topics in Part 2

- Code Signing
- Understanding Permissions
- Application Sandbox
- Filesystem Encryption

The Security Model

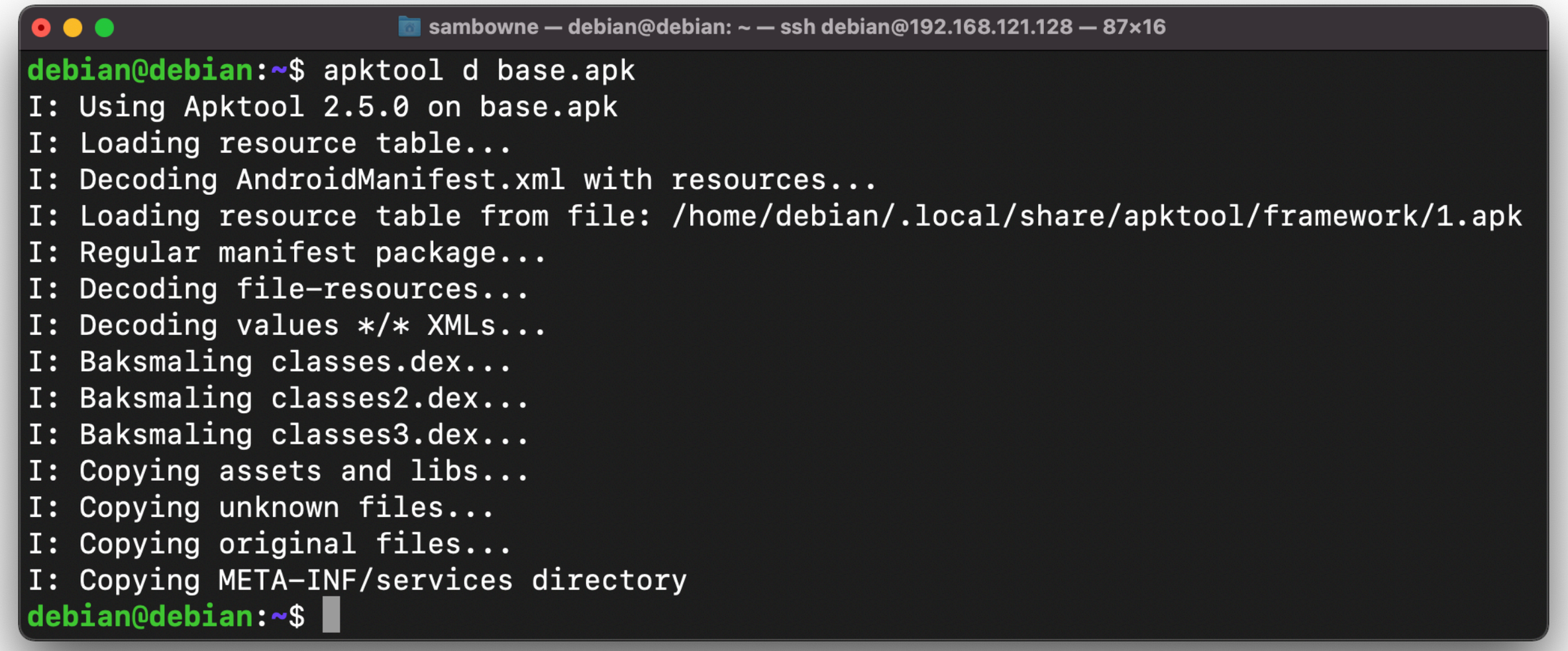
- No app should be able to access another app's data without authorization
- Open and extensible environment
- Android must know who created an app
  - At least to know whether Google made it or not

Code Signing

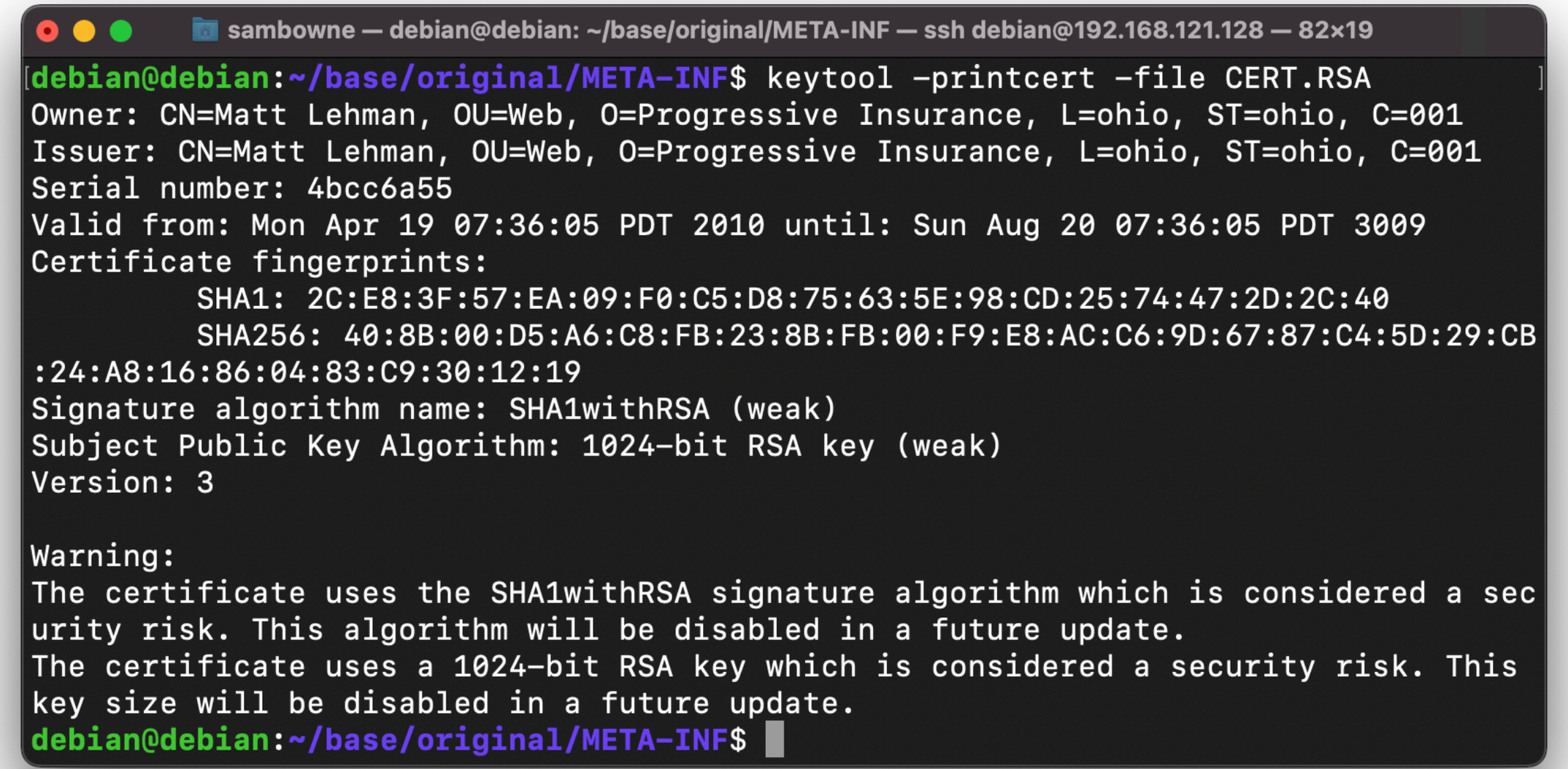
Digital Certificates

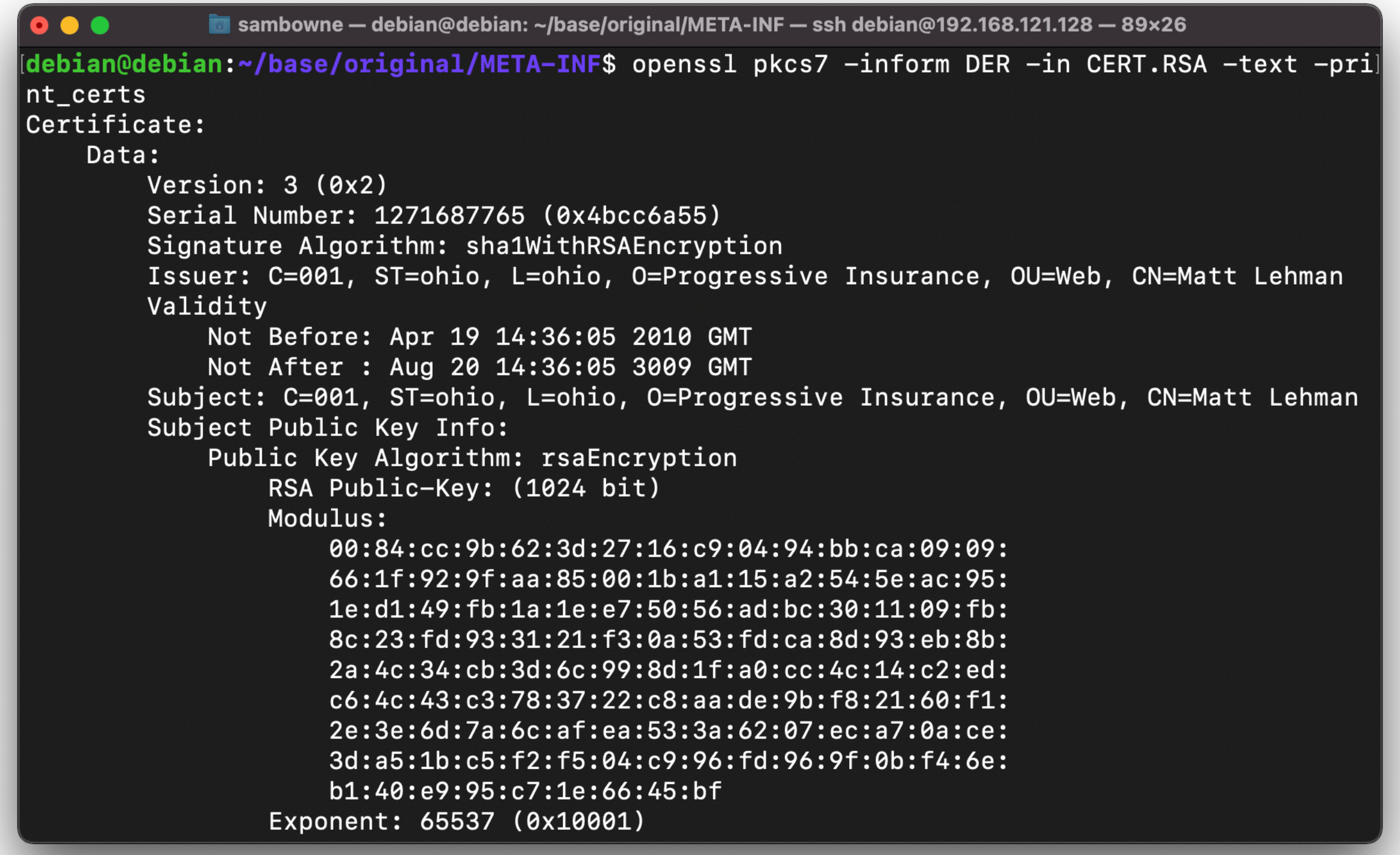
- Public-key cryptography
- Private key held only by app developer
- Generate key with **keytool**
- Sign app with **jarsigner**
- Signature in META-INF directory

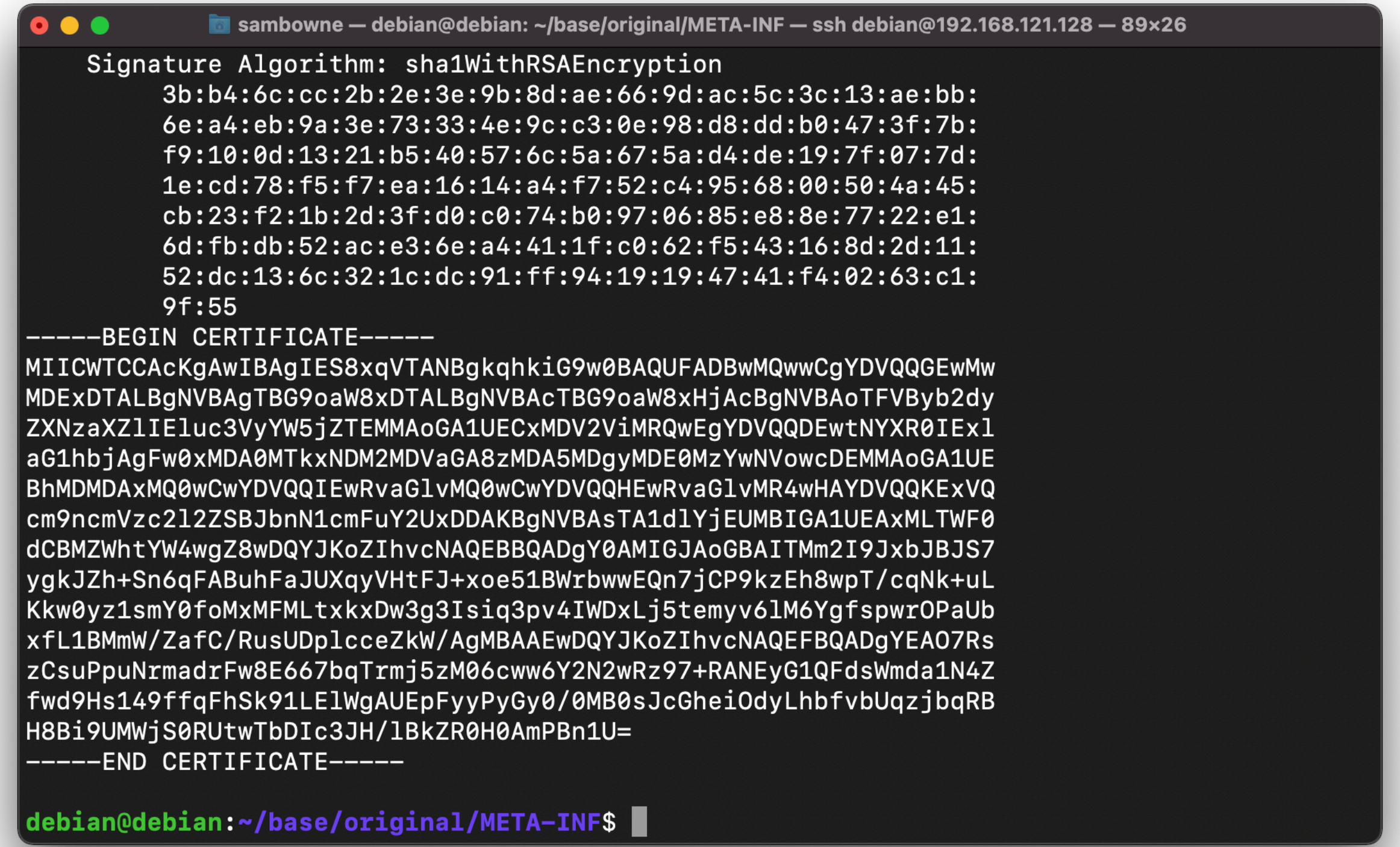
Unpacking the Progressive App



The Certificate in META-INF




sambowne — debian@debian: ~/base/original/META-INF — ssh debian@192.168.121.128 — 89×26
debian@debian:~/base/original/META-INF$ openssl pkcs7 -inform DER -in CERT.RSA -text -pri
nt_certs
Certificate:
Data:
Version: 3 (0x2)
Serial Number: 1271687765 (0x4bcc6a55)
Signature Algorithm: sha1WithRSAEncryption
Issuer: C=001, ST=ohio, L=ohio, O=Progressive Insurance, OU=Web, CN=Matt Lehman
Validity
Not Before: Apr 19 14:36:05 2010 GMT
Not After : Aug 20 14:36:05 3009 GMT
Subject: C=001, ST=ohio, L=ohio, O=Progressive Insurance, OU=Web, CN=Matt Lehman
Subject Public Key Info:
Public Key Algorithm: rsaEncryption
RSA Public-Key: (1024 bit)
Modulus:
00:84:cc:9b:62:3d:27:16:c9:04:94:bb:ca:09:09:
66:1f:92:9f:aa:85:00:1b:a1:15:a2:54:5e:ac:95:
1e:d1:49:fb:1a:1e:e7:50:56:ad:bc:30:11:09:fb:
8c:23:fd:93:31:21:f3:0a:53:fd:ca:8d:93:eb:8b:
2a:4c:34:cb:3d:6c:99:8d:1f:a0:cc:4c:14:c2:ed:
c6:4c:43:c3:78:37:22:c8:aa:de:9b:f8:21:60:f1:
2e:3e:6d:7a:6c:af:ea:53:3a:62:07:ec:a7:0a:ce:
3d:a5:1b:c5:f2:f5:04:c9:96:fd:96:9f:0b:f4:6e:
b1:40:e9:95:c7:1e:66:45:bf
Exponent: 65537 (0x10001)


sambowne — debian@debian: ~/base/original/META-INF — ssh debian@192.168.121.128 — 89×26
Signature Algorithm: sha1WithRSAEncryption
3b:b4:6c:cc:2b:2e:3e:9b:8d:ae:66:9d:ac:5c:3c:13:ae:bb:
6e:a4:eb:9a:3e:73:33:4e:9c:c3:0e:98:d8:dd:b0:47:3f:7b:
f9:10:0d:13:21:b5:40:57:6c:5a:67:5a:d4:de:19:7f:07:7d:
1e:cd:78:f5:f7:ea:16:14:a4:f7:52:c4:95:68:00:50:4a:45:
cb:23:f2:1b:2d:3f:d0:c0:74:b0:97:06:85:e8:8e:77:22:e1:
6d:fb:db:52:ac:e3:6e:a4:41:1f:c0:62:f5:43:16:8d:2d:11:
52:dc:13:6c:32:1c:dc:91:ff:94:19:19:47:41:f4:02:63:c1:
9f:55
-----BEGIN CERTIFICATE-----
MIICWTCCAcKgAwIBAgIES8xqVTANBgkqhkiG9w0BAQUFADBwMQwwCgYDVQQGEwMw
MDExDTALBgNVBAgTBG9oaW8xDTALBgNVBAcTBG9oaW8xHjAcBgNVBAoTFVByb2dy
ZXNzaXZlIEluc3VyYW5jZTEMMAoGA1UECxMDV2ViMRQwEgYDVQQDEwtNYXR0IExl
aG1hbjAgFw0xMDA0MTkxNDM2MDVaGA8zMDA5MDgyMDE0MzYwNVowcDEMMAoGA1UE
BhMDMDAxMQ0wCwYDVQQIEwRvaGlvMQ0wCwYDVQQHEwRvaGlvMR4wHAYDVQQKExVQ
cm9ncmVzc2l2ZSBJbnN1cmFuY2UxDDAKBgNVBAsTA1dlYjEUMBIGA1UEAxMLTWF0
dCBMZWhtYW4wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAITMm2I9JxbJBJS7
ygkJZh+Sn6qFABuhFaJUXqyVHtFJ+xoe51BWrbwwEQn7jCP9kzEh8wpT/cqNk+uL
Kkw0yz1smY0foMxMFMLtxkxDw3g3Isiq3pv4IWDxLj5temyv6lM6YgfspwrOPaUb
xfL1BMmW/ZafC/RusUDplcceZkW/AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAO7Rs
zCsuPpuNrmadrFw8E667bqTrmj5zM06cww6Y2N2wRz97+RANEyG1QFdsWmda1N4Z
fwd9Hs149ffqFhSk91LElWgAUEpFyyPyGy0/0MB0sJcGheiOdyLhbfvbUqzjbqRB
H8Bi9UMWjS0RUtwTbDIc3JH/lBkZR0H0AmPBn1U=
-----END CERTIFICATE-----
debian@debian:~/base/original/META-INF$

MD5 Collisions

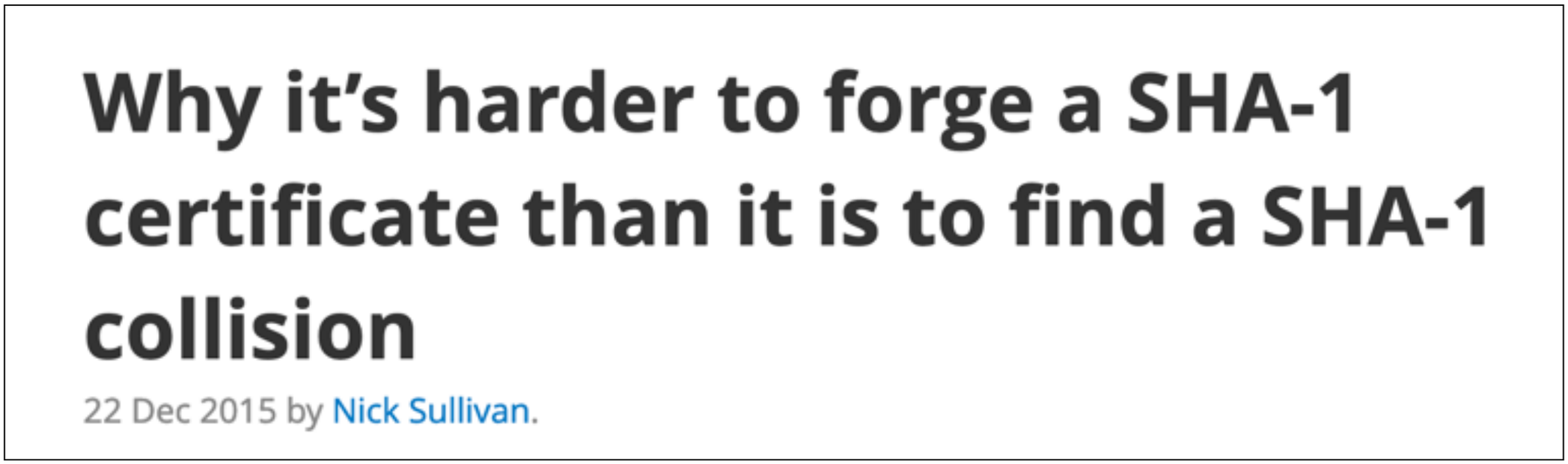
SSL broken! Hackers create rogue CA certificate using MD5 collisions

Using computing power from a cluster of 200 PS3 game consoles and about $700 in test digital certificates, a group of hackers in the U.S.

By Ryan Naraine for Zero Day | December 30, 2008 -- 06:00 GMT (22:00 PST) | Topic: Enterprise Software

- Link Ch 6a

Collision Attack Unlikely



Why it's harder to forge a SHA-1 certificate than it is to find a SHA-1 collision

22 Dec 2015 by Nick Sullivan.

- Link Ch 6d

Certificate Validation

- Android does not verify the certificate in any way
- Certificates don't need to come from a trusted Certificate Authority
- Most are self-signed
- Certificate checked only when app is installed

Certificate Validity Period

- Google recommends a valid period of 25 years or longer
  - So you can update your app

Signing Vulnerabilities

- Master Key
- "Extra" Field Length
- "Name" Field Length

Master Key

- Found in 2013 by BlueBox Security
- If two files are in the APK archive with the same filenames
  - Only the first file's hash is checked
  - But the second file is actually deployed to the device
- Arbitrary code execution possible

"Extra" Field Length

- Length field is a 16-bit value
  - Java treats it as a **signed integer**
- Can overflow and become negative
- Allows injection of altered files that pass signature verification

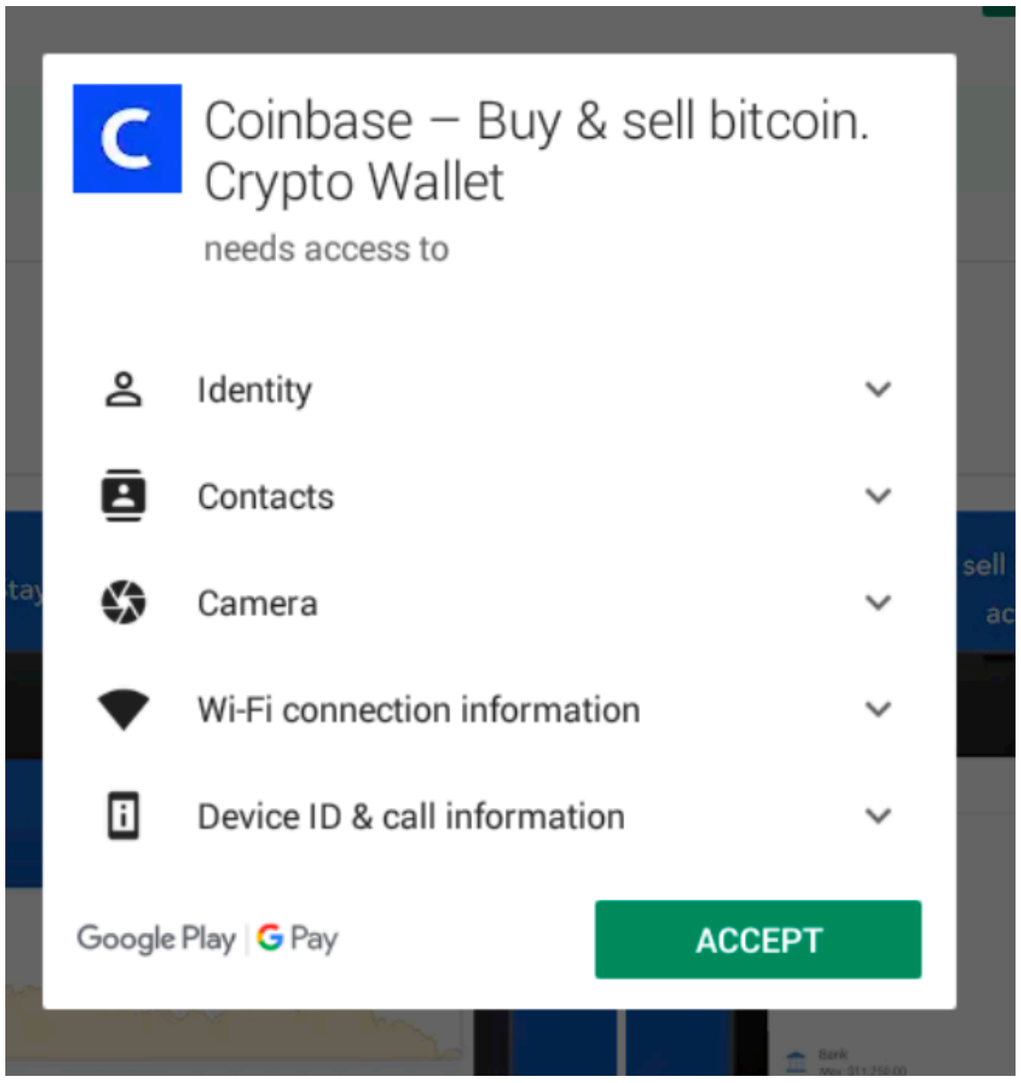
"Name" Field Length

- Length not checked by the Java verification code
- Allows code injection into the filename
- While passing signature validation

Understanding Permissions

The Android Permission Model

- Permissions shown at install time



AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8" standalone="no"?><manifest xmlns:android="http://schemas.android.com/apk
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" android:required="true"/>
    <permission android:name="com.infonow.bofa.permission.C2D_MESSAGE" android:protectionLevel="signature"/>
    <uses-permission android:name="android.permission.GET_ACCOUNTS" android:required="true"/>
    <uses-permission android:name="com.infonow.bofa.permission.C2D_MESSAGE"/>
    <uses-permission android:name="com.infonow.bofa.com.google.android.c2dm.permission.RECEIVE"/>
    <uses-permission android:name="android.permission.WAKE_LOCK" android:required="true"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE" android:required="true"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" android:required="true"/>
    <uses-permission android:name="android.permission.CALL_PHONE"/>
    <uses-feature android:name="android.hardware.location.gps"/>
    <uses-permission android:name="android.permission.INTERNET" android:required="true"/>
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" android:required="true"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" android:required="true"/>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
    <uses-permission android:name="android.net.conn.CONNECTIVITY_CHANGE"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_UPDATES"/>
    <uses-permission android:name="android.permission.CAMERA"/>
```

Permission Protection Levels

- An app can define a new permission
- When it does, a **protection level** is assigned to it
- Skype defines this permission

```
<permission android:name=
"com.skype.raider.permission.C2D_MESSAGE"
android:protectionLevel="signature"/>
```

Permission Protection Levels

"**`normal`**"	The default value. A lower-risk permission that gives requesting applications access to isolated application-level features, with minimal risk to other applications, the system, or the user. The system automatically grants this type of permission to a requesting application at installation, without asking for the user's explicit approval (though the user always has the option to review these permissions before installing).
"**`dangerous`**"	A higher-risk permission that would give a requesting application access to private user data or control over the device that can negatively impact the user. Because this type of permission introduces potential risk, the system may not automatically grant it to the requesting application. For example, any dangerous permissions requested by an application may be displayed to the user and require confirmation before proceeding, or some other approach may be taken to avoid the user automatically allowing the use of such facilities.

Permission Protection Levels

"**`signature`**"	A permission that the system grants only if the requesting application is signed with the same certificate as the application that declared the permission. If the certificates match, the system automatically grants the permission without notifying the user or asking for the user's explicit approval.
"**`signatureOrSystem`**"	*Old synonym for* "**`signature|privileged`**". *Deprecated in API level 23.* A permission that the system grants only to applications that are in a dedicated folder on the Android system image *or* that are signed with the same certificate as the application that declared the permission. Avoid using this option, as the "**`signature`**" protection level should be sufficient for most needs and works regardless of exactly where apps are installed. The "**`signatureOrSystem`**" permission is used for certain special situations where multiple vendors have applications built into a system image and need to share specific features explicitly because they are being built together.

Permission Protection Levels

- **system**
  - Part ot the Android system image
  - Or app installed in some folders on the **/system** partition
- **development**
  - Permissions applied at runtime
  - Uncommon, poorly documented

"Signature" Protection

- Recommended for apps that don't intend to share data or functionality with apps from other developers
- No other apps can access your app's components

Malicious Apps

- Can just ask for permissions and hope the user allows it (social engineering)
- Or include a kernel exploit to gain root, such as Gingerbreak

Application Sandbox

Data Folder Permissions

- Each app runs as its own user
- Unless it requests to run as **sharedUserId** and has the same signature as another app
- Some apps allow world-execute, like Schwab

```
vbox86p:/data/data # ls -l | egrep "wab|coinb"

drwx------  8 u0_a145         u0_a145         4096 2019-02-04 17:25 com.coinbase.android
drwxr-x--x  7 u0_a129         u0_a129         4096 2019-01-25 10:41 com.schwab.mobile.ssm
vbox86p:/data/data #
```

Sandbox Limitations

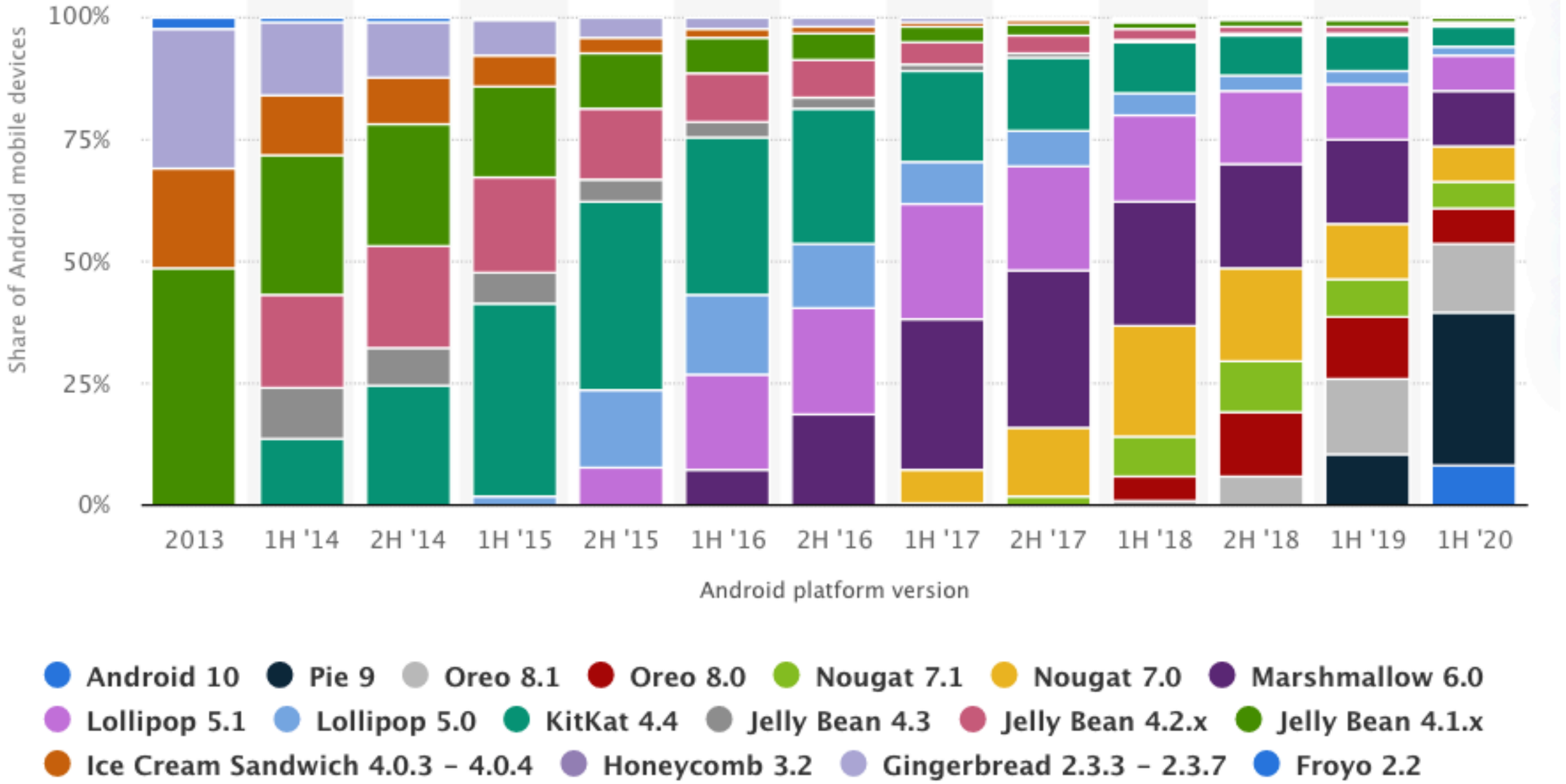
- Not a separate virtual machine for each app
- Only Linux user and group permissions

Filesystem Encryption

"Full Disk Encryption"

- Prevents data theft from a stolen device
- Available since Android v. 3.0
- Not enabled by default in versions prior to 5.0
- Encrypts with AES-CBC, a strong algorithm
- FDE is going away, replaced by file-based encryption (link Ch 6a)

Android Versions



- Link Ch 6b

Encryption Limitations

- SD card not encrypted
- Only protects data at rest
- If attacker can execute code on the device, encryption does nothing

6b