# Introduction to Mobile Security Testing

## Approaches and Examples using OWASP MSTG

OWASP German Day 20.11.2018

Carlos Holguera

# $ whoami

**Carlos Holguera** [olˈɣera]

- Security Engineer working at ESCRYPT GmbH since 2012
- Area of expertise:
  - Mobile & Automotive Security Testing
  - Security Testing Automation

@grepharder

# Index

# 1 Why?

# Why?

Online videos,
articles,
trainings ??

- Trustworthy sources?
- Right Methodology?
- Latest Techniques?

✓ MASVS is the WHAT

✓ MSTG is the HOW

# 2 From the Standard to the Guide

# From the Standard to the Guide

# From the Standard to the Guide

## OWASP Mobile Application Security Verification Standard

This is the official Github Repository of the OWASP Mobile Application Security Verification Standard (MASVS). The MASVS establishes baseline security requirements for mobile apps that are useful in many scenarios, including:

- In the SDLC - to establish security requirements to be followed by solution architects and developers;
- In mobile app penetration tests - to ensure completeness and consistency in mobile app penetration tests;
- In procurement - as a measuring stick for mobile app security, e.g. in form of questionnaire for vendors;
- Et cetera.

The MASVS is a sister project of the OWASP Mobile Security Testing Guide.


### Getting the MASVS

PDF downloads are available on the Releases page. The current release is MASVS version 1.1. The MASVS is also available in different languages:

- Spanish
- Russian

Also as Word, ePub, JSON, CSV, XML, ... see the README

Open on GitHub

Read it on GitBook

# From the Standard to the Guide

OWASP Mobile Application Security Verification Standard

OS agnostic



## Security Verification Requirements

| # | Description | L1 | L2 |
|---|---|---|---|
| **5.1** | Data is encrypted on the network using TLS. The secure channel is used consistently throughout the app. | ✓ | ✓ |
| **5.2** | The TLS settings are in line with current best practices, or as close as possible if the mobile operating system does not support the recommended standards. | ✓ | ✓ |
| **5.3** | The app verifies the X.509 certificate of the remote endpoint when the secure channel is established. Only certificates signed by a trusted CA are accepted. | ✓ | ✓ |
| **5.4** | The app either uses its own certificate store, or pins the endpoint certificate or public key, and subsequently does not establish connections with endpoints that offer a different certificate or key, even if signed by a trusted CA. | | ✓ |
| **5.5** | The app doesn't rely on a single insecure communication channel (email or SMS) for critical operations, such as enrollments and account recovery. | | ✓ |
| **5.6** | The app only depends on up-to-date connectivity and security libraries. | | ✓ |

How? MSTG

# From the Standard to the Guide

## OWASP Mobile Application Security Verification Standard

**Mobile Application Security Requirements - Android**

| ID | Detailed Verification Requirement | Level 1 | Level 2 | Status | Testing Procedure |
|---|---|---|---|---|---|
| **V1** | **Architecture, design and threat modelling** | | | | |
| 1.1 | Verify all application components are identified and are known to be needed. | ✓ | ✓ | Pass | - |
| 1,2 | Verify that security controls are never enforced only on the client side, but on the respective remote endpoints. | ✓ | ✓ | | - |
| 1,3 | Verify that a high-level architecture for the mobile app and all connected remote services has been defined and security has been addressed in that architecture. | ✓ | ✓ | | - |
| 1,4 | Verify that data considered sensitive in the context of the mobile app is clearly identified. | ✓ | ✓ | | - |
| 1,5 | Verify all app components are defined in terms of the business functions and/or security functions they provide. | | ✓ | N/A | - |
| 1,6 | Verify that a threat model for the mobile app and the associated remote services, which identifies potential threats and countermeasures, has been produced. | | ✓ | N/A | - |
| 1.7 | Verify that all security controls have a centralized implementation. | | ✓ | N/A | - |
| 1,8 | Verify that there is an explicit policy for how cryptographic keys (if any) are managed, and the lifecycle of cryptographic keys is enforced. Ideally, follow a key management standard such as NIST SP 800-57. | | ✓ | N/A | - |
| 1,9 | Verify that a mechanism for enforcing updates of the mobile app exists. | | ✓ | N/A | - |
| 1.10 | Verify that security is addressed within all parts of the software development lifecycle. | | ✓ | N/A | - |
| **V2** | **Data Storage and Privacy** | | | | |
| 2.1 | Verify that system credential storage facilities are used appropriately to store sensitive data, such as user credentials or cryptographic keys. | ✓ | ✓ | | Testing For Sensitive Data in Local Data Storage |
| 2.2 | Verify that no sensitive data is written to application logs. | ✓ | ✓ | | Testing For Sensitive Data in Logs |
| 2.3 | Verify that no sensitive data is shared with third parties unless it is a necessary part of the architecture. | ✓ | ✓ | | Testing Whether Sensitive Data Is Sent To Third Parti |
| 2.4 | Verify that the keyboard cache is disabled on text inputs that process sensitive data. | ✓ | ✓ | | Testing Whether the Keyboard Cache Is Disabled for |
| 2.5 | Verify that the clipboard is deactivated on text fields that may contain sensitive data. | ✓ | ✓ | | Testing for Sensitive Data in the Clipboard |
| 2.6 | Verify that no sensitive data is exposed via IPC mechanisms. | ✓ | ✓ | | Testing Whether Sensitive Data Is Exposed via IPC Me |
| 2.7 | Verify that no sensitive data, such as passwords or pins, is exposed through the user interface. | ✓ | ✓ | | Testing for Sensitive Data Disclosure Through the Use |
| 2.8 | Verify that no sensitive data is included in backups generated by the mobile operating system. | | ✓ | N/A | Testing for Sensitive Data in Backups |
| 2.9 | Verify that the app removes sensitive data from views when backgrounded. | | ✓ | N/A | Testing for Sensitive Information in Auto-Generated |
| 2.10 | Verify that the app does not hold sensitive data in memory longer than necessary, and memory is cleared explicitly after use. | | ✓ | N/A | Testing for Sensitive Data in Memory |
| 2.11 | Verify that the app enforces a minimum device-access-security policy, such as requiring the user to set a device passcode. | | ✓ | N/A | Testing the Device-Access-Security Policy |

Get from GitHub

fork & customize
dep. on target

# From the Standard to the Guide

## OWASP Mobile Security Testing Guide

Thanks Berndt
& Sven for starting this!



We do have a message to our readers however! The first rule of the OWASP Mobile Security Testing Guide is: Don't just follow the OWASP Mobile Security Testing Guide. True excellence at mobile application security requires a deep understanding of mobile operating systems, coding, network security, cryptography, and a whole lot of other things, many of which we can only touch on briefly in this book. Don't stop at security testing. Write your own apps, compile your own kernels, dissect mobile malware, learn how things tick. And as you keep learning new things, consider contributing to the MSTG yourself! Or, as they say: "Do a pull request".

Summit Team


Open on GitHub

Read it on GitBook

Also as Word, leanpub,
PFD, ... see the README

# From the Standard to the Guide

OWASP Mobile Security Testing Guide

GitHub Search or clone & `grep`

MASVS Refs. on each chapter

## Android Network APIs

**References**

**OWASP Mobile Top 10 2016**

- M3 - Insecure Communication - https://ww
Insecure_Communication

**OWASP MASVS**

- V5.3: "The app verifies the X.509 certificat
established. Only certificates signed by a t
- V5.4: "The app either uses its own certifica
subsequently does not establish connectic
even if signed by a trusted CA."
- V5.6: "The app only depends on up-to-date connectivity and security libraries."

lore Marketplace Pricing keychain / Sign in or Sign up

22 code results in OWASP/owasp-mstg or view all results on GitHub

Sort: Best match ▾

Document/0x06b-Basic-Security-Testing.md

Markdown

Showing the top two matches Last indexed 20 days ago

```
$ scp -P 2222 root@localhost:/tmp/data.tgz .
```

```

#### Dumping KeyChain Data

[Keychain-dumper](https://github.com/ptoomey3/Keychain-Dumper/) lets you dump a jailbroken device's KeyChain contents. The easiest way to get the tool is to download the binary from its GitHub repo:
```

Document/0x06e-Testing-Cryptography.md

Markdown

Showing the top two matches Last indexed 20 days ago

```
Next, for asymmetric operations, Apple provides [SecKey](https://opensource.apple.com/source/Secur
57740.51.3/keychain/SecKey.h.auto.html "SecKey"). Apple provides a nice guide in its [Developer Doc
(https://developer.apple.com/documentation/security/certificate_key_and_trust_services/keys/using_k
"Using keys for encryption") on how to use this.
...
*Source: https://stackoverflow.com/questions/8569555/pbkdf2-using-commoncrypto-on-ios, tested in
the `Arcane` library*

When you need to store the key, it is recommended to use the Keychain as long as the protection cl
`kSecAttrAccessibleAlways`. Storing keys in any other location, such as the `NSUserDefaults`, Prope
```

# 3 Vulnerability Analysis

# Vulnerability Analysis

## Static Analysis (SAST)

### Manual Code Review

- `grep` & line-by-line examination
- **expert** code reviewer proficient in both language and frameworks

### Automatic Code Analysis

- Speed up the review
- Predefined set of rules or industry best practices
- False positives! A **security professional** must always review the results.
- False negatives! Even worse …

## Dynamic Analysis (DAST)

### Testing and evaluation of apps

- Real-time execution
- Manual
- Automatic

### Examples of checks

- disclosure of data in transit
- authentication and authorization issues
- server configuration errors.

Recommendation: SAST + DAST + security professional

# Vulnerability Analysis

Based on MASVS

### Static Analysis

Check the app's source code for logging mechanisms by searching for the following keywords:

- Functions and classes, such as:
  - `android.util.Log`
  - `Log.d` | `Log.e` | `Log.i` | `Log.v` | `Log.w` |
  - `Logger`
- Keywords and system output:
  - `System.out.print` | `System.err.print`
  - logfile
  - logging
  - logs

While preparing the production release, you can use tools
delete logging-related code. To determine whether all the
have been removed, check the ProGuard configuration file

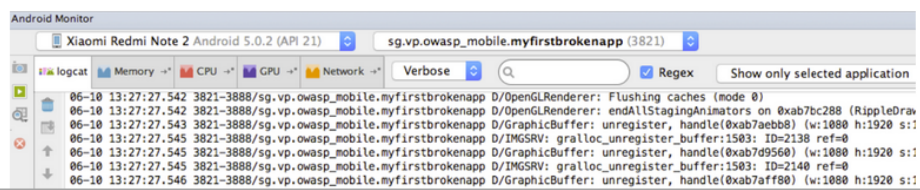
```
-assumenosideeffects class android.util.Log
```

### Dynamic Analysis

Use all the mobile app functions at least once, then identify the application's data directory and look for log files ( `/data/data/<package-name>` ). Check the application logs to determine whether log data has been generated; some mobile applications create and store their own logs in the data directory.

Many application developers still use `System.out.println` or `printStackTrace` instead of a proper logging class. Therefore, your testing strategy must include all output generated while the application is starting, running and closing. To determine what data is directly printed by `System.out.println` or `printStackTrace`, you can use `Logcat`. There are two ways to execute Logcat:

- Logcat is part of *Dalvik Debug Monitor Server* (DDMS) and Android Studio. If the app is running in debug mode, the log output will be shown in the Android Monitor on the Logcat tab. You can filter the app's log output by defining patterns in Logcat.




* OWASP, Mobile Security Testing Guide, 2018 (0x05d-Testing-Data-Storage.html)

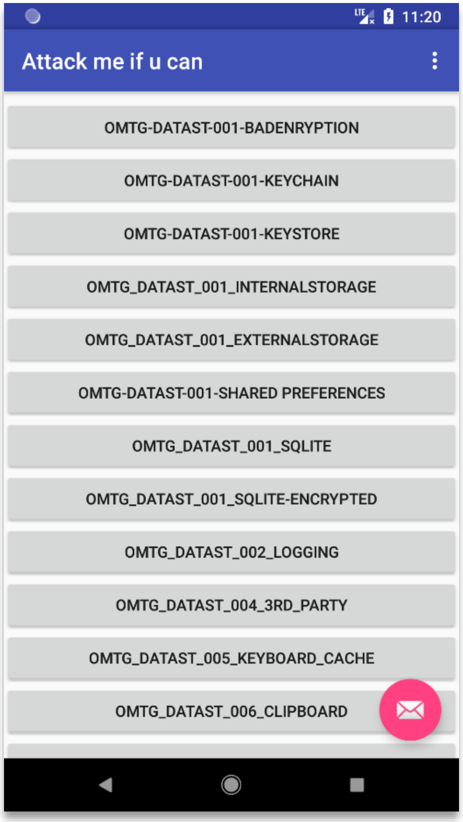
What to verify & how.
Incl. References to
MASVS Requirements

# Vulnerability Analysis

Demo App

## The MSTG Hacking Playground App

Open on GitHub



# Vulnerability Analysis

## Manual Code Review

Example: Android **original** source code

```java
    // Using Java-AES-Crypto, https://github.com/tozny/java-aes-crypto
    public void decryptString() {
        // BTW: Really bad idea, as this is the raw private key. Should be stored in the keystore
        String rawKeys = "4zInk+d4jlQ3m1B1ELctxg==:4aZtzwpbniebvM7yC4/GIa2ZmJpSzqrAFtVk91Rm+Q4=";
        AesCbcWithIntegrity.SecretKeys privateKey = null;
        try {
            privateKey = AesCbcWithIntegrity.keys(r
        } catch (InvalidKeyException e) {
            e.printStackTrace();
        }

        String cipherTextString = "6WpfZkgKMJsPhHNhWoSpVg==:6/TgUCXrAuAa2lUMPWhx8hHOWjWEHFp3VIsz3Ws37ZU=:C0mWyNQjcf6n7eBSFz

        AesCbcWithIntegrity.CipherTextIvMac cipherTextIvMac = new AesCbcWithIntegrity.CipherTextIvMac(cipherTextString);
        try {
            plainText = AesCbcWithIntegrity.decryptString(cipherTextIvMac, privateKey);
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
```

**8.11** All executable files and libraries belonging to the app are either encrypted on the file level and/or important code and data segments inside the executables are encrypted or packed. Trivial static analysis does not reveal important code or data.

# Vulnerability Analysis

## Manual Code Review

Example: Android **decompiled** source code

```
     */
    public void decryptString() {
        String string2 = "4zInk+d4jlQ3m1B1ELctxg==:4aZtzwpbniebvM7yC4/GIa2ZmJpSzqrAFtVk91Rm+Q4=";
        AesCbcWithIntegrity$SecretKeys aesCbcWithIntegrity$SecretKeys = null;
        try {
            aesCbcWithIntegrity$SecretKeys = AesCbcWithIntegrity.keys(string2);
        }
        catch (InvalidKeyException invalidKeyException) {
            invalidKeyException.printStackTrace();
        }
        String string3 = "6WpfZkgKMJsPhHNhWoSpVg==:6/TgUCXrAuAa2lUMPWhx8hHOWjWEHFp3VIsz3Ws37ZU=:C0mWyNQjcf
        AesCbcWithIntegrity$CipherTextIvMac aesCbcWithIntegrity$CipherTextIvMac = new AesCbcWithIntegrity$
        try {
            String string4;
            this.plainText = string4 = AesCbcWithIntegrity.decryptString(aesCbcWithIntegrity$CipherTextIvMa
            return;
        }
```

**8.12** If the goal of obfuscation is to protect sensitive computations, an obfuscation scheme is used that is both appropriate for the particular task and robust against manual and automated de-obfuscation methods, considering currently published research. The effectiveness of the obfuscation scheme must be verified through manual testing. Note that hardware-based isolation features are preferred over obfuscation whenever possible.

# Vulnerability Analysis

## Manual Code Review

Example: iOS **original** source code

```
- (void)storeCredentialsInKeychain {
    NSMutableDictionary *storeCredentials = [NSMutableDictionary dictionary];

    // Prepare keychain dict for storing credentials.
    [storeCredentials setObject:(id)CFBridgingRelease(kSecClassGenericPassword) forKey:(id)CFBridgingRelease(kSecClass)];

    // Store password encoded.
    [storeCredentials setObject:[self.password.text dataUsingEncoding:NSUTF8StringEncoding] forKey:(id)CFBridgingRelease(kSecValueD
    [storeCredentials setObject:self.username.text forKey:(id)CFBridgingRelease(kSecAttrAccount)];

    // Access keychain data for this app, only when unlocked. Imp to have this while
    // adding as well as updating keychain item. This is the default, but best practice
    // to specify if apple changes its API.
    [storeCredentials setObject:(id)CFBridgingRelease(kSecAttrAccessibleWhenUnlocked) forKey:(id)CFBridgingRelease(kSecAttrAccessib

    // Query Keychain to see if credentials exist.
    OSStatus results = SecItemCopyMatching((CFDictionaryRef) CFBridgingRetain(storeCredentials), nil);

    // If username exists in keychain...
    if (results == errSecSuccess) {
        // NSDictionary *dataFromKeyChain = NULL;
        CFDataRef dataFromKeyChain;

        // There will always be one matching entry, thus limit resultset size to 1.
        [storeCredentials setObject:(id)CFBridgingRelease(kSecMatchLimitOne) forKey:(id)CFBridgingRelease(kSecMatchLimit)];
```

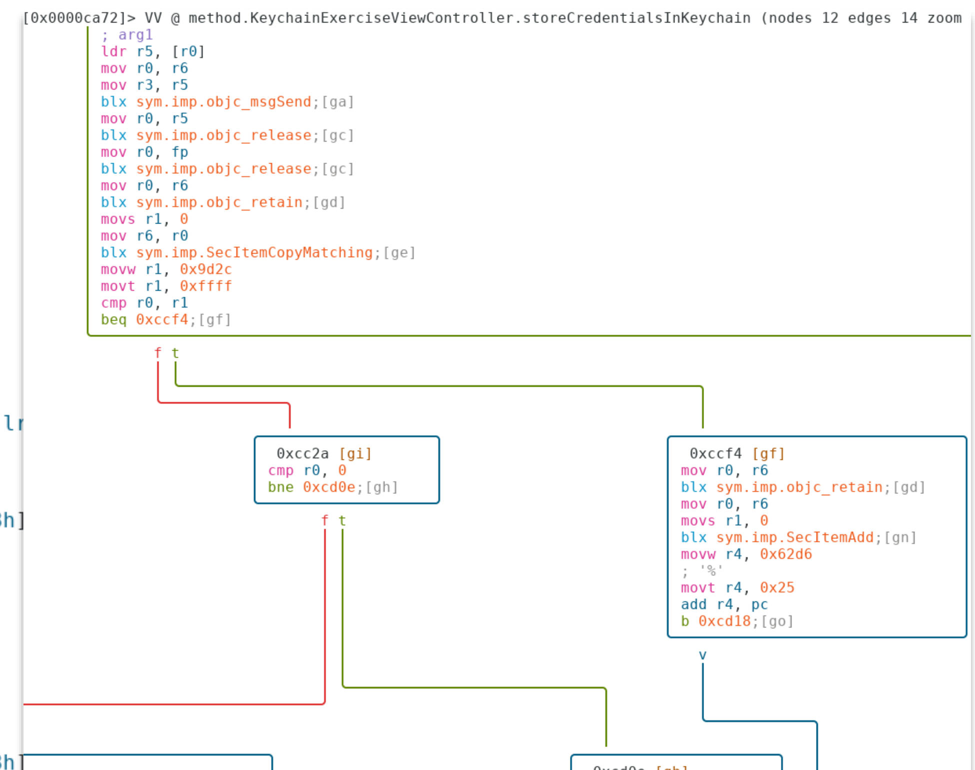
* OWASP iGoat A Learning Tool for iOS App Pentesting and Security, 2018 (iGoat)

# Vulnerability Analysis

## Manual Code Review

Example: iOS **disassembled** “source code”

```
(fcn) method.KeychainExerciseViewController.storeCredentialsInKeychain 1202
  method.KeychainExerciseViewController.storeCredentialsInKeychain (int arg1, int arg2, int arg3, int arg4);
          ; var int local_0h @ sp+0x0
          ; var int local_4h @ sp+0x4
          ; var int local_10h @ sp+0x10
          ; var int local_14h @ sp+0x14
          ; var int local_18h @ sp+0x18
          ; var int local_1ch @ sp+0x1c
          ; var int local_20h @ sp+0x20
          ; var int local_24h @ sp+0x24
          ; var int local_28h @ sp+0x28
          ; var int local_2ch @ sp+0x2c
          ; var int local_30h @ sp+0x30
          ; var int local_34h @ sp+0x34
          ; var int local_38h @ sp+0x38
          ; var int local_3ch @ sp+0x3c
          ; arg int arg1 @ r0
          ; arg int arg2 @ r1
          ; arg int arg3 @ r2
          ; arg int arg4 @ r3
          0x0000ca72      f0b5          push {r4, r5, r6, r7, lr
          0x0000ca74      03af          add r7, sp, 0xc
          0x0000ca76      2de9000d      push.w {r8, sl, fp}
          0x0000ca7a      91b0          sub sp, 0x44
          0x0000ca7c      0e90          str r0, [sp + local_38h]
          0x0000ca7e      8046          mov r8, r0
          0x0000ca80      46f6ac20      movw r0, 0x6aac
          0x0000ca84      c0f22800      movt r0, 0x28
          0x0000ca88      4af2d632      movw r2, 0xa3d6
          0x0000ca8c      c0f22802      movt r2, 0x28
          0x0000ca90      7844          add r0, pc
          0x0000ca92      7a44          add r2, pc
          0x0000ca94      0168          ldr r1, [r0]
          0x0000ca96      1068          ldr r0, [r2]
          0x0000ca98      0a91          str r1, [sp + local_28h]
          0x0000ca9a      e8f164ec      blx sym.imp.objc_msgSend    ;[1]
```



# Vulnerability Analysis

## Automatic Code Analysis

### Example: Static Analyzer

**+ SSL Connection Checking**

URLs that are NOT under SSL (Total:1):

http://xmlpull.org/v1/doc/features.html#process-namespaces

=> Lcom/mwr/example/sieve/DBParser;->getPIN(Ljava/io/InputStream;)Ljava/lang/String;

=> Lcom/mwr/example/sieve/DBParser;->getKey(Ljava/io/InputStream;)Ljava/lang/String;

=> Lcom/mwr/example/sieve/DBParser;->readFile(Ljava/io/InputStream;)Ljava/util/List;

**+ SSL Certificate Verification Checking**

This app DOES NOT check the validation of SSL Certificate. It allows self-signed, expired or mismatch CN certificates for SSL connection.

This is a critical vulnerability and allows attackers to do MITM attacks without your knowledge.

If you are transmitting users' username or password, these sensitive information may be leaking.

Reference:

(1)OWASP Mobile Top 10 doc: https://www.owasp.org/index.php/Mobile_Top_10_2014-M3

(2)Android Security book: http://goo.gl/BFb65r

(3)https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=134807561

This vulnerability is much more severe than Apple's "goto fail" vulnerability: http://goo.gl/eFlovw

Please do not try to create a "X509Certificate" and override "checkClientTrusted", "checkServerTrusted", and "getAcceptedIssuers" functions with blank implementation.

We strongly suggest you use the existing API instead of creating your own X509Certificate class.

Please modify or remove these vulnerable code:

[Confirm Vulnerable]

must be always evaluated
by a professional

# 4 Information Gathering

# Information Gathering

## Information Gathering

### Identifies

- General Information
- Sensitive Information

... on the target that is publically available. E.g. about the OS and its APIs

### Evaluates the risk by understanding

- Existing Vulnerabilities
- Existing Exploits

... especially from third party software.

# Information Gathering

## Android Platform Overview

This section introduces the Android platform from the architecture point of view. The following four key areas are discussed:

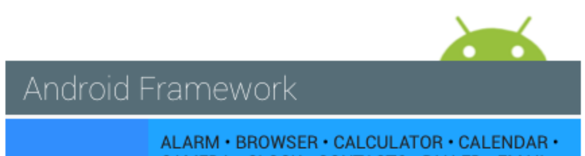
1. Android security architecture
2. Android application structure
3. Inter-process Communication (IPC)
4. Android application publishing

Visit the official Android developer documentation website for more details about the Android platform.

## Android Security Architecture

Android is a Linux-based open source platform developed by Google as a mobile operating system (OS). Today the platform is the foundation for a wide variety of modern technology, such as mobile phones, tablets, wearable tech, TVs, and other "smart" devices. Typical Android builds ship with a range of pre-installed ("stock") apps and support installation of third-party apps through the Google Play store and other marketplaces.

Android's software stack is composed of several different layers. Each layer defines interfaces and offers specific services.




* OWASP, Mobile Security Testing Guide, 2018 (0x05a-Platform-Overview.html)

# Information Gathering

Example: Open OMTG_DATAST_011_Memory.java and observe the *decryptString* implementation.

```
    // Using Java-AES-Crypto, https://github.com/tozny/java-aes-crypto
    public void decryptString() {
        // BTW: Really bad idea, as this is the raw private key. Should be stored in the keystore
        String rawKeys = "4zInk+d4jlQ3m1B1ELctxg==:4aZtzwpbniebvM7yC4/GIa2ZmJpSzqrAFtVk91Rm+Q4=";
        AesCbcWithIntegrity.SecretKeys privateKey = null;
        try {
            privateKey = AesCbcWithIntegrity.keys(rawKeys);
        } catch (InvalidKeyException e) {
            e.printStackTrace();
        }

        String cipherTextString = "6WpfZkgKMJsPhHNhWoSpVg==:6/TgUCXrAuAa21UMPWhx8hHOWjWEHFp3VIsz3Ws37ZU=:C0mWyNQjcf6n7eBSFzmkXqxdu55CjUOIc5qFw0

        AesCbcWithIntegrity.CipherTextIvMac cipherTextIvMac = new AesCbcWithIntegrity.CipherTextIvMac(cipherTextString);
        try {
            plainText = AesCbcWithIntegrity.decryptString(cipherTextIvMac, privateKey);
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
```

# Information Gathering

Let me google that for you…

com.tozny.crypto.android.AesCbcWithIntegrity;

Alle News Videos Shopping Bilder Mehr Einstellungen Tools

Ungefähr 2.400 Ergebnisse (0,37 Sekunden)

java-aes-crypto/AesCbcWithIntegrity.java at master · tozny/java-aes ...
https://github.com/tozny/...crypto/...crypto/.../com/tozny/crypto/a... ▾ Diese Seite übersetzen
12.11.2014 - java-aes-crypto/aes-crypto/src/main/java/**com/tozny/crypto/android/**
**AesCbcWithIntegrity**.java .... private static final String CIPHER = "AES";.

GitHub - tozny/java-aes-crypto: A simple Android class for encrypting ...
https://github.com/tozny/java-aes-crypto ▾ Diese Seite übersetzen
A simple **Android** class for encrypting & decrypting strings, aiming to avoid the .... **CipherTextIvMac**

## How to include in project?

### Copy and paste

It's a single very simple java class, AesCbcWithIntegrity.java that works across most or all versions of Android. The class should be easy to paste into an existing codebase.

# Information Gathering

:. (US) https://github.com/tozny/java-aes-crypto/blob/master/aes-crypto/src/main/java/com/tozny/crypto/android com.tozny.crypto.android.AesCbcWi

```java
public class AesCbcWithIntegrity {
    // If the PRNG fix would not succeed for some reason, we normally will throw an exception.
    // If ALLOW_BROKEN_PRNG is true, however, we will simply log instead.
    private static final boolean ALLOW_BROKEN_PRNG = false;

    private static final String CIPHER_TRANSFORMATION = "AES/CBC/PKCS5Padding";
    private static final String CIPHER = "AES";
    private static final int AES_KEY_LENGTH_BITS = 128;
    private static final int IV_LENGTH_BYTES = 16;
    private static final int PBE_ITERATION_COUNT = 10000;
    private static final int PBE_SALT_LENGTH_BITS = AES_KEY_LENGTH_BITS; // same size as key output
    private static final String PBE_ALGORITHM = "PBKDF2WithHmacSHA1";

    //Made BASE_64_FLAGS public as it's useful to know for compatibility.
    public static final int BASE64_FLAGS = Base64.NO_WRAP;
    //default for testing
    static final AtomicBoolean prngFixed = new AtomicBoolean(false);

    private static final String HMAC_ALGORITHM = "HmacSHA256";
    private static final int HMAC_KEY_LENGTH_BITS = 256;

```

Got all original crypto code inclusive crypto params.

# 5 Penetration Testing

# Penetration Testing

## Preparation

### Coordination with the client

- Define **scope** / focus
- Request source code
- Release and debug apps
- Understand **customer worries**

### Identifying Sensitive Data

- at rest: file
- in use: address space
- in transit: tx to endpoint, IPC

## Intelligence Gathering

### Environmental info

- Goals and **intended use** (e.g. Flashlight)
- What if compromised?

### Architectural Info

- Runtime protections (jailbreak, emulator..?)
- Which OS (old versions?)
- Network Security
- Secure Storage (what, why, how?)

# Penetration Testing

## Mapping

Based on all previous information

- UNDERSTAND the target
- LIST potential vulnerabilities
- DRAW sensitive data flow
- DESIGN a **test plan**, use **MASVS**

Complement with automated scanning and manually exploring the app

## Exploitation

- Exploit the vulnerabilities identified during the previous phase
- Use the **MSTG**
- Find the **true positives**

## Reporting

- Essential to the client
- Not so fun?
- It makes you the *bad guy*
- Security not integrated early enough in the SDLC?

# Penetration Testing

### Penetration Testing (a.k.a. Pentesting)

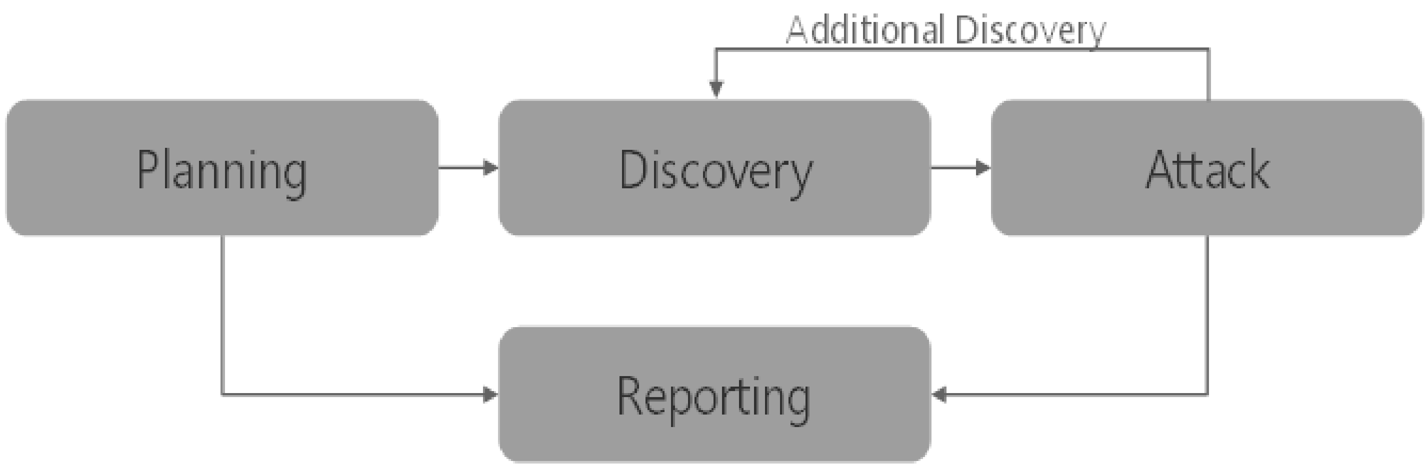
The classic approach involves all-around security testing of the app's final or near-final build, e.g., the build that's available at the end of the development process. For testing at the end of the development process, we recommend the Mobile App Security Verification Standard (MASVS) and the associated checklist. A typical security test is structured as follows:

- **Preparation** - defining the scope of security testing, including identifying applicable security controls, the organization's testing goals, and sensitive data. More generally, preparation includes all synchronization with the client as well as legally protecting the tester (who is often a third party). Remember, attacking a system without written authorization is illegal in many parts of the world!
- **Intelligence Gathering** - analyzing the **environmental** and **architectural** context of the app to gain a general contextual understanding.
- **Mapping the Application** - based on information from the previous phases; may be complemented by automated scanning and manually exploring the app. Mapping provides a thorough understanding of the app, its entry points, the data it holds, and the main potential vulnerabilities. These vulnerabilities can then be ranked according to the damage their exploitation would cause so that the security tester can prioritize them. This phase includes the creation of test cases that may be used during test execution.
- **Exploitation** - in this phase, the security tester tries to penetrate the app by exploiting the vulnerabilities identified during the previous phase. This phase is necessary for determining whether vulnerabilities are real (i.e., true positives).
- **Reporting** - in this phase, which is essential to the client, the security tester reports the vulnerabilities he or she has been able to exploit and documents the kind of compromise he or she has been able to perform, including the compromise's scope (for example, the data he or she has been able to access illegitimately).

* OWASP, Mobile Security Testing Guide, 2018 (0x04b-Mobile-App-Security-Testing.html)

# Penetration Testing

Penetration Testing is conducted in four phases*




* NIST, Technical Guide to Information Security Testing and Assessment, 2008

# Penetration Testing

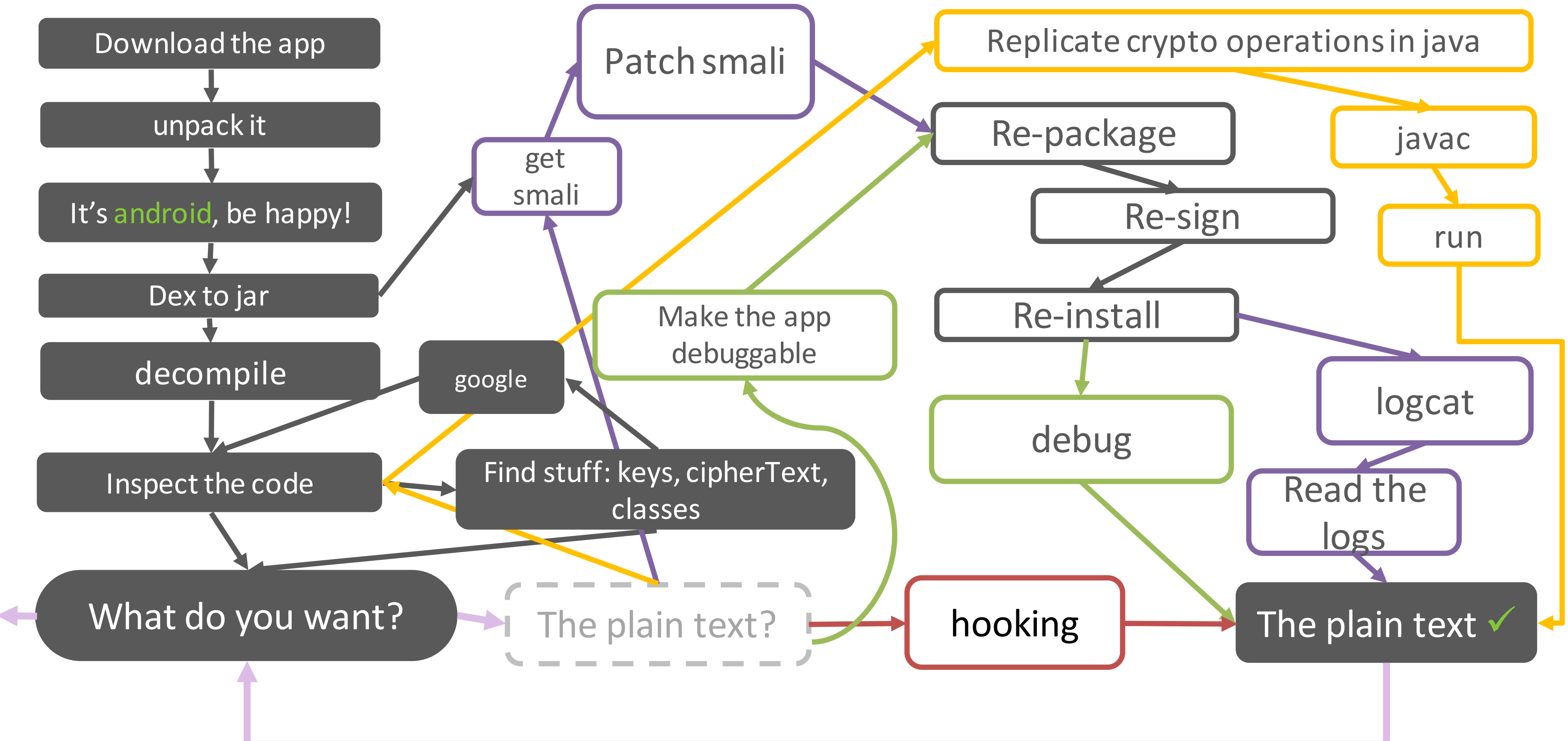
However

- ☞ Multiple attack vectors
- ☞ Multiple steps
- ☞ Different combinations give different full attack vectors

*So penetration testing usually looks more like this …*

# Penetration Testing

## Demo Spoiler



# Penetration Testing

## Techniques

decompilation

fuzzing

traffic interception

method tracing

code injection

tampering

disassembly

hooking

traffic dump

root detection

man-in-the-middle

dynamic binary instrumentation

debugging

binary patching

# Penetration Testing

All techniques and testing methods categorized

ANDROID TESTING GUIDE

- Platform Overview
- Setting up a Testing Environment for Android Apps
- Testing Data Storage on Android
- Android Cryptographic APIs
- Local Authentication on Android
- Android Network APIs
- Android Platform APIs
- Code Quality and Build Settings for Android Apps
- Tampering and Reverse Engineering on Android
- Android Anti-Reversing Defenses

IOS TESTING GUIDE

- Platform Overview
- Setting up a Testing Environment for iOS Apps
- Data Storage on iOS
- iOS Cryptographic APIs
- Local Authentication on iOS
- iOS Network APIs
- iOS Platform APIs
- Code Quality and Build Settings for iOS Apps
- Tampering and Reverse Engineering on iOS
- iOS Anti-Reversing Defenses

One for Android,
one for iOS. All happy ☺

# Penetration Testing

## Reverse Engineering

Reverse engineering is the process of taking an app apart to find out how it works. You can do this by examining the compiled app (static analysis), observing the app during run time (dynamic analysis), or a combination of both.

### Statically Analyzing Java Code

Java bytecode can be converted back into source code without many problems unless some nasty, tool-breaking anti-decompilation tricks have been applied. We'll be using UnCrackable App for Android Level 1 in the following examples, so download it if you haven't already. First, let's install the app on a device or emulator and run it to see what the crackme is about.

```
$ wget https://github.com/OWASP/owasp-mstg/raw/master/Crackmes/Android/Level_01/UnCrackable-Level1
$ adb install UnCrackable-Level1.apk
```

* OWASP, Mobile Security Testing Guide, 2018 (0x05c-Reverse-Engineering-and-Tampering.html)

# Penetration Testing

## Tampering and Runtime Instrumentation

First, we'll look at some simple ways to modify and instrument mobile apps. *Tampering* means making patches or run-time changes to the app to affect its behavior. For example, you may want to deactivate SSL pinning or binary protections that hinder the testing process. *Runtime Instrumentation* encompasses adding hooks and runtime patches to observe the app's behavior. In mobile app-sec however, the term loosely refers to all kinds of run-time manipulation, including overriding methods to change behavior.

### Patching and Re-Packaging

Making small changes to the app Manifest or bytecode is often the quickest way to fix small annoyances that prevent you from testing or reverse engineering an app. On Android, two issues in particular happen regularly:

1. You can't attach a debugger to the app because the android:debuggable flag is not set to true in the Manifest.
2. You can't intercept HTTPS traffic with a proxy because the app employs SSL pinning.

In most cases, both issues can be fixed by making minor changes to the app and then re-signing and re-packaging it. Apps that run additional integrity checks beyond default Android code-signing are an exception—in these cases, you have to patch the additional checks as well.

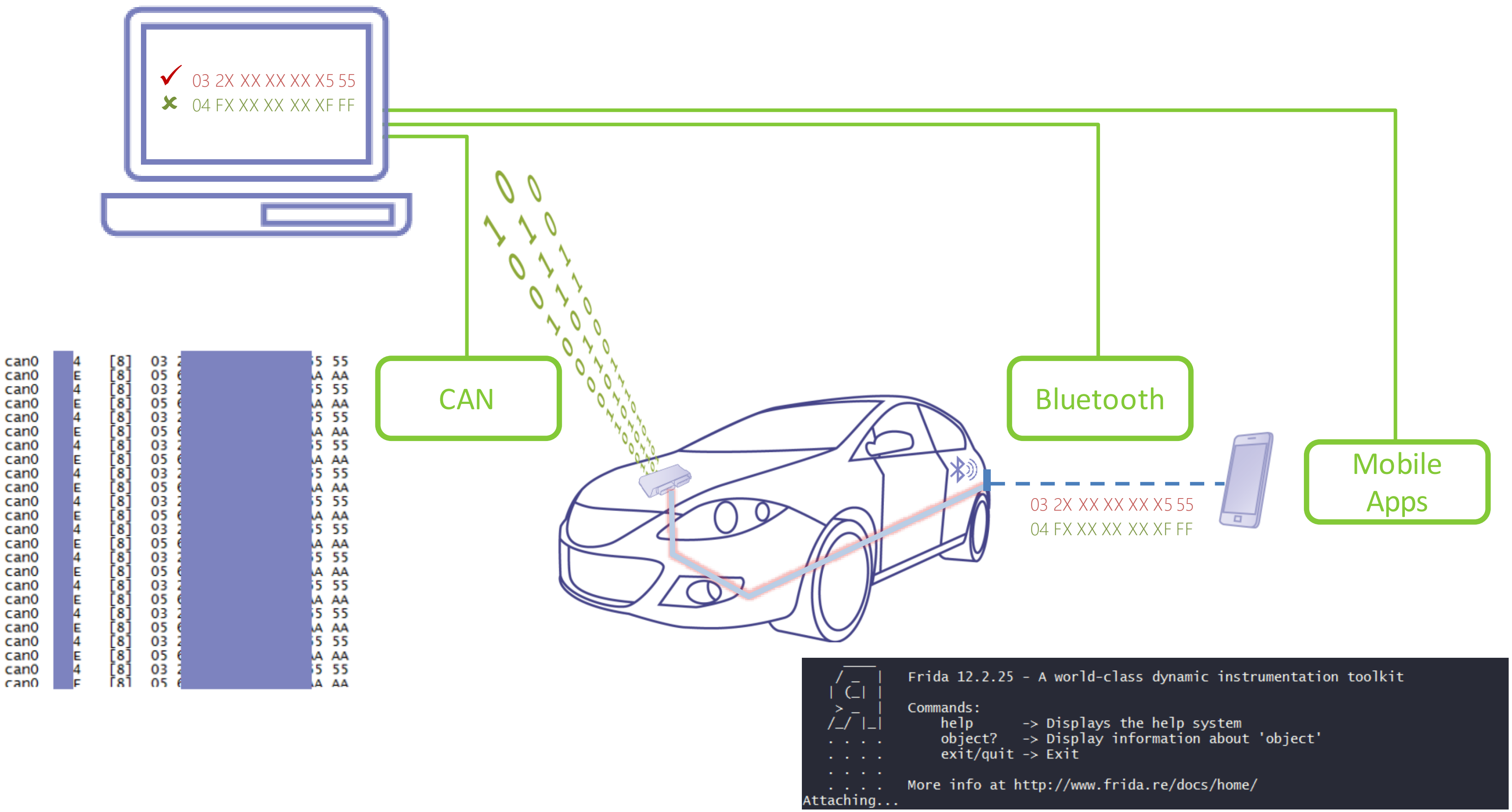
#### Example: Disabling Certificate Pinning

Certificate pinning is an issue for security testers who want to intercept HTTPS communication for

[*] OWASP, Mobile Security Testing Guide, 2018 (0x05c-Reverse-Engineering-and-Tampering.html)

# Penetration Testing

## Example Scenario Automotive-Mobile Testing



# 6 Demo 1 Mobile Penetration Testing

Let‘s decrypt that encrypted string!

# Demo 1

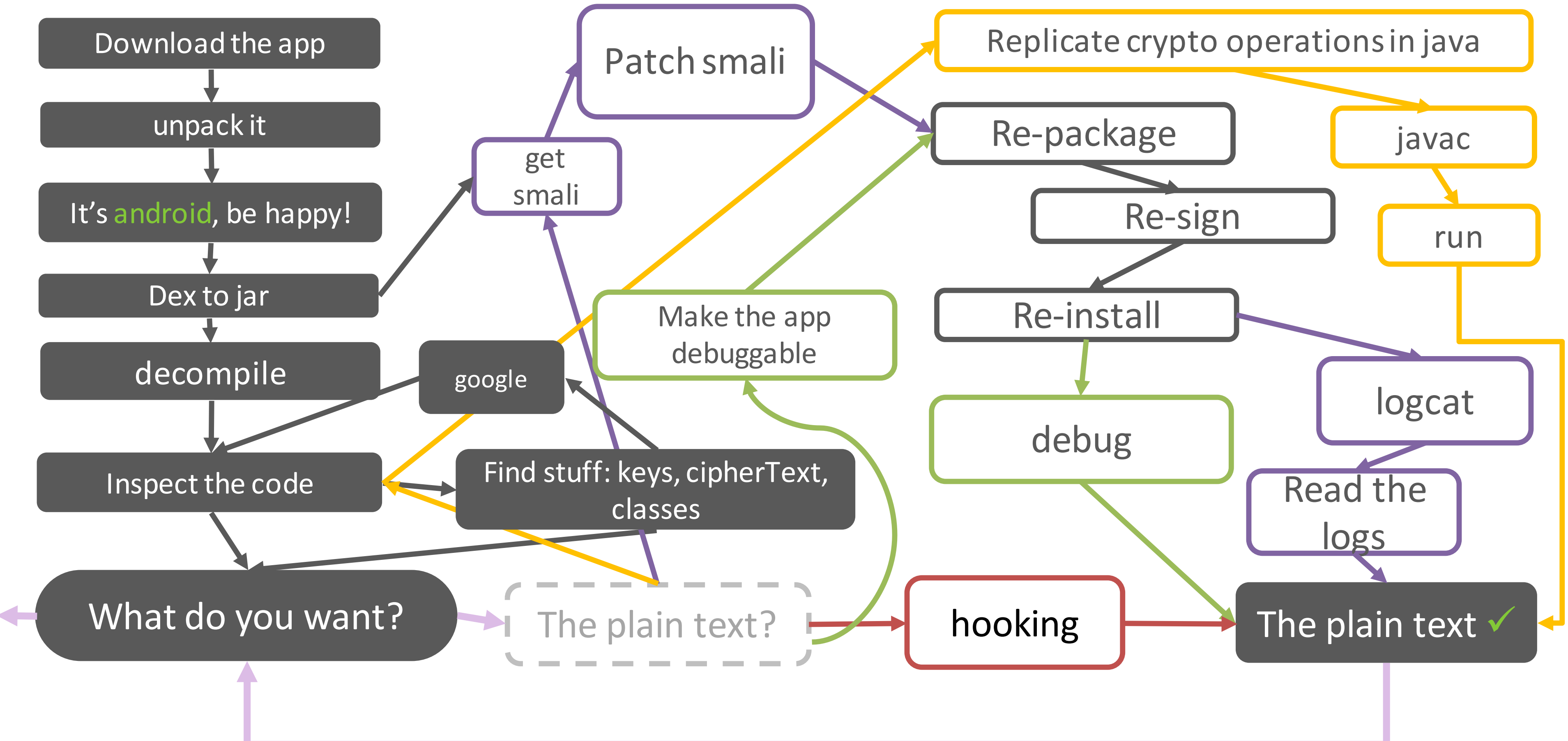
## App: MSTG-Hacking-Playground (011_MEMORY)

```java
    // Using Java-AES-Crypto, https://github.com/tozny/java-aes-crypto
    public void decryptString() {
        // BTW: Really bad idea, as this is the raw private key. Should be stored in the keystore
        String rawKeys = "4zInk+d4jlQ3m1B1ELctxg==:4aZtzwpbniebvM7yC4/GIa2ZmJpSzqrAFtVk91Rm+Q4=";
        AesCbcWithIntegrity.SecretKeys privateKey = null;
        try {
            privateKey = AesCbcWithIntegrity.keys(rawKeys);
        } catch (InvalidKeyException e) {
            e.printStackTrace();
        }

        String cipherTextString = "6WpfZkgKMJsPhHNhWoSpVg==:6/TgUCXrAuAa21UMPWhx8hHOWjWEHFp3VIsz3Ws37ZU=:C0mWyNQjcf6n7eBSFzmkXqxdu55CjUOIc5qFw0

        AesCbcWithIntegrity.CipherTextIvMac cipherTextIvMac = new AesCbcWithIntegrity.CipherTextIvMac(cipherTextString);
        try {
            plainText = AesCbcWithIntegrity.decryptString(cipherTextIvMac, privateKey);
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
```

We have the keys and the ciphertext

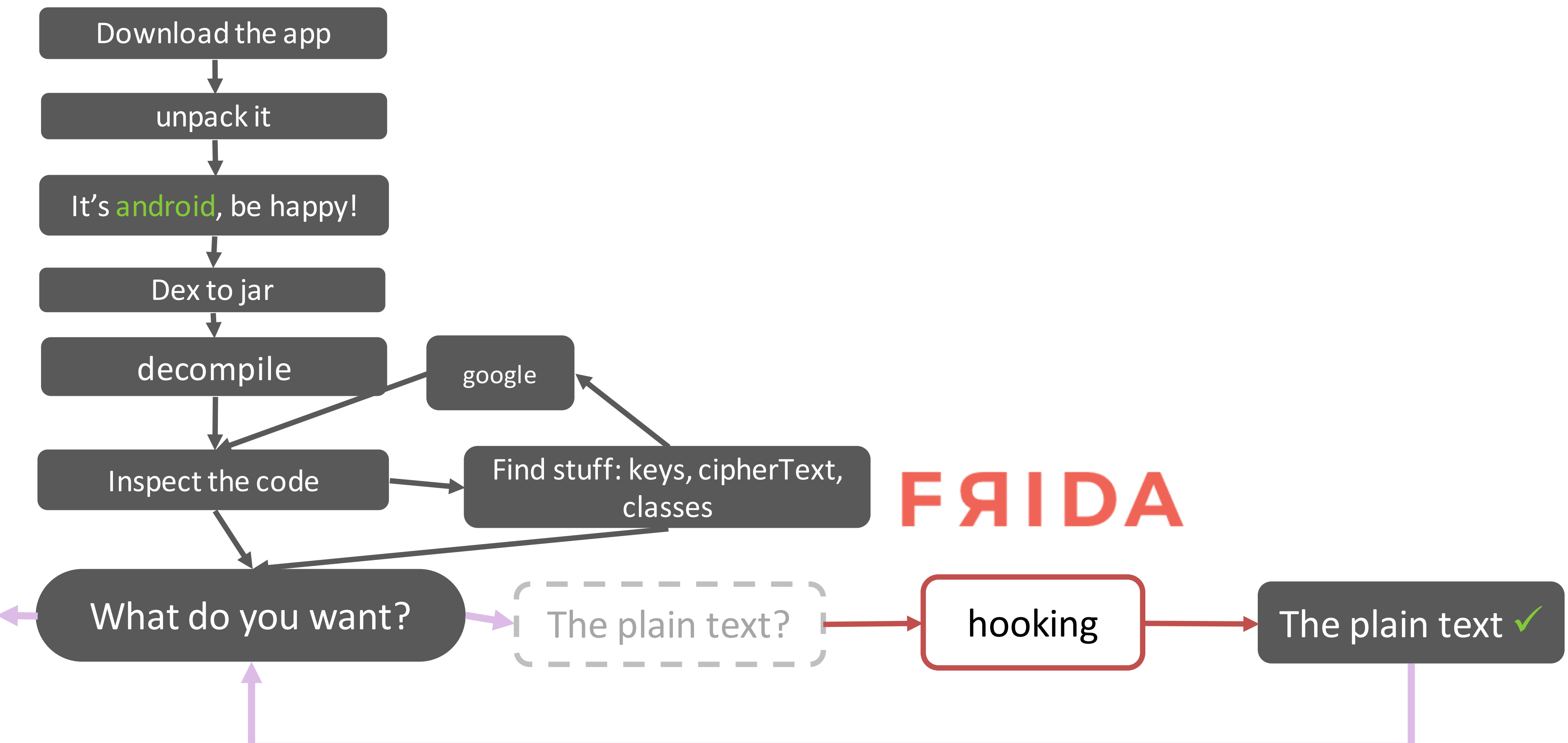
But the plaintext remains inside this variable. And now?

# Demo 1



Download the app
unpack it
It's android, be happy!
Dex to jar
decompile
Inspect the code
google
Find stuff: keys, cipherText, classes
get smali
Patch smali
Make the app debuggable
Replicate crypto operations in java
javac
run
Re-package
Re-sign
Re-install
debug
logcat
Read the logs
What do you want?
The plain text?
hooking
The plain text ✓

# Demo 1

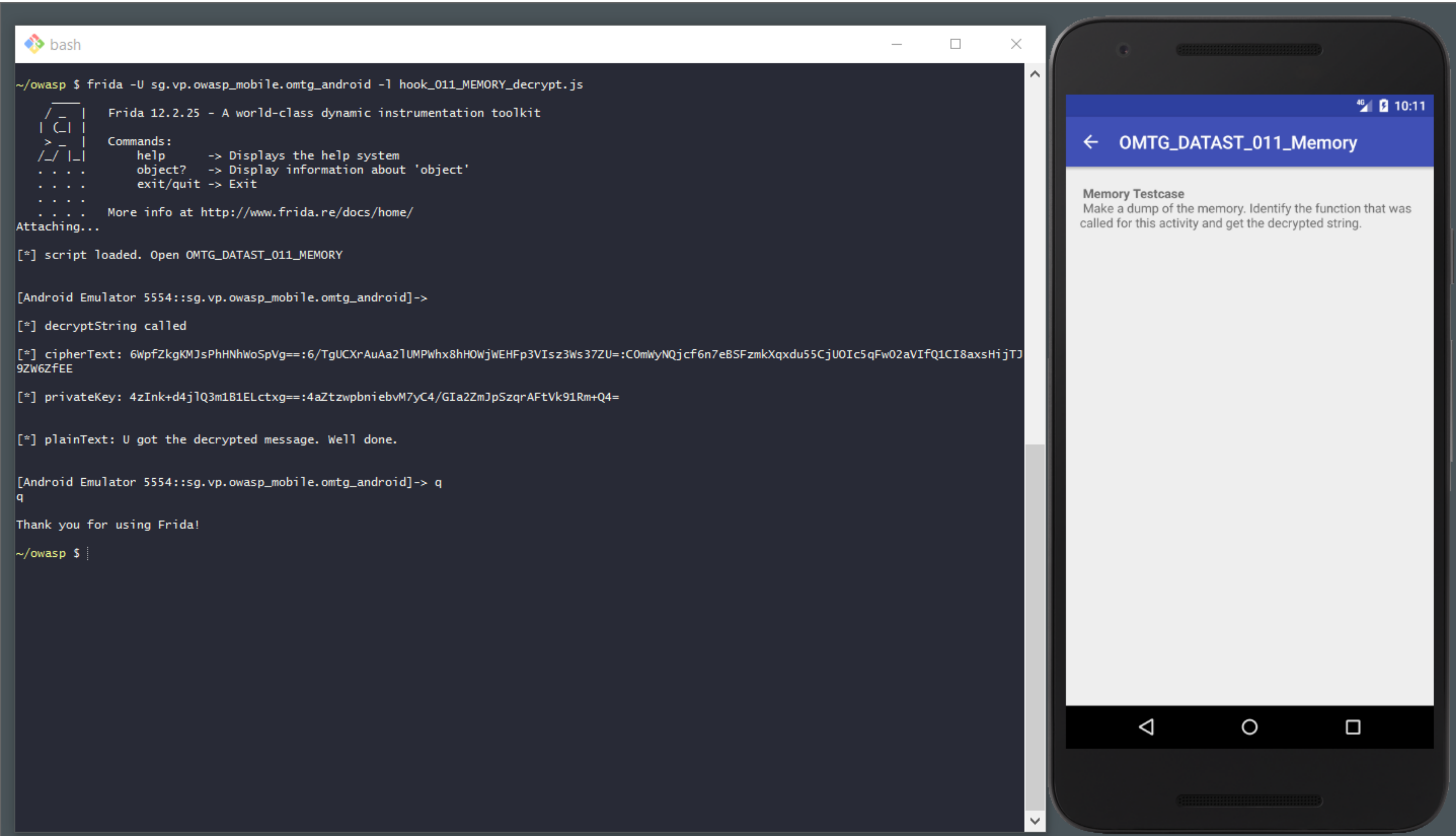


Download the app
unpack it
It's android, be happy!
Dex to jar
decompile
google
Inspect the code
Find stuff: keys, cipherText, classes
FRIDA
What do you want?
The plain text?
hooking
The plain text ✓

# Demo 1

```
Java.perform(function(){
  console.log("\n[*] script loaded. Open OMTG_DATAST_011_MEMORY\n\n");
  var clazz = Java.use("com.tozny.crypto.android.AesCbcWithIntegrity");

  clazz.decryptString.overload('com.tozny.crypto.android.AesCbcWithIntegrity$CipherTextIvMac', 'com.tozny.crypto.androi
.implementation = function (cipherText, privateKey) {
    console.log("\n\n[*] decryptString called");
    console.log("\n[*] cipherText: " + cipherText);
    console.log("\n[*] privateKey: " + privateKey);

    var ret = this.decryptString.overload('com.tozny.crypto.android.AesCbcWithIntegrity$CipherTextIvMac', 'com.tozny.cr
.call(this, cipherText, privateKey);
    console.log('\n\n[*] plainText: ' + ret);
    return ret;
  };
});
```

# Demo 1



bash
~/owasp $ frida -U sg.vp.owasp_mobile.omtg_android -l hook_011_MEMORY_decrypt.js
Frida 12.2.25 - A world-class dynamic instrumentation toolkit
Commands:
help -> Displays the help system
object? -> Display information about 'object'
exit/quit -> Exit
More info at http://www.frida.re/docs/home/
Attaching...
[*] script loaded. Open OMTG_DATAST_011_MEMORY
[Android Emulator 5554::sg.vp.owasp_mobile.omtg_android]->
[*] decryptString called
[*] cipherText: 6WpfZkgKMJsPhHNhWoSpVg==:6/TgUCXrAuAa2lUMPWhx8hHOWjWEHFp3VIsz3Ws37ZU=:COmWyNQjcf6n7eBSFzmkXqxdu55CjUOIc5qFwO2aVIfQ1CI8axsHijTJ9ZW6ZfEE
[*] privateKey: 4zInk+d4jlQ3m1B1ELctxg==:4aZtzwpbniebvM7yC4/GIa2ZmJpSzqrAFtVk91Rm+Q4=
[*] plainText: U got the decrypted message. Well done.
[Android Emulator 5554::sg.vp.owasp_mobile.omtg_android]-> q
q
Thank you for using Frida!
~/owasp $
10:11
OMTG_DATAST_011_Memory
Memory Testcase
Make a dump of the memory. Identify the function that was called for this activity and get the decrypted string.

# 6 Demo 2 Mobile Penetration Testing

Let‘s get the crypto keys!

# Demo 2

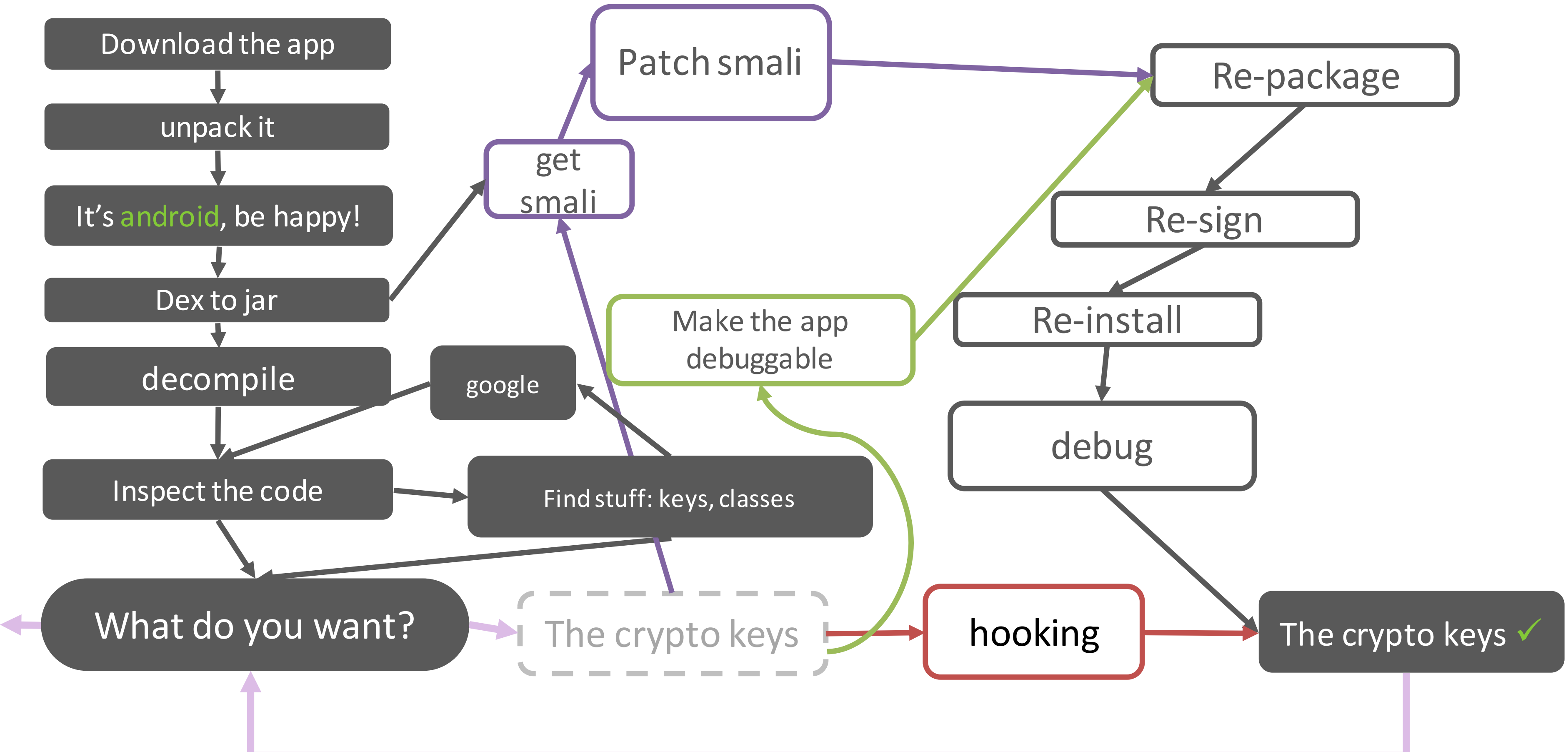
App: MSTG-Hacking-Playground (001_KEYSTORE)

### Extraction prevention

Key material of Android Keystore keys is protected from extraction using two security measures:

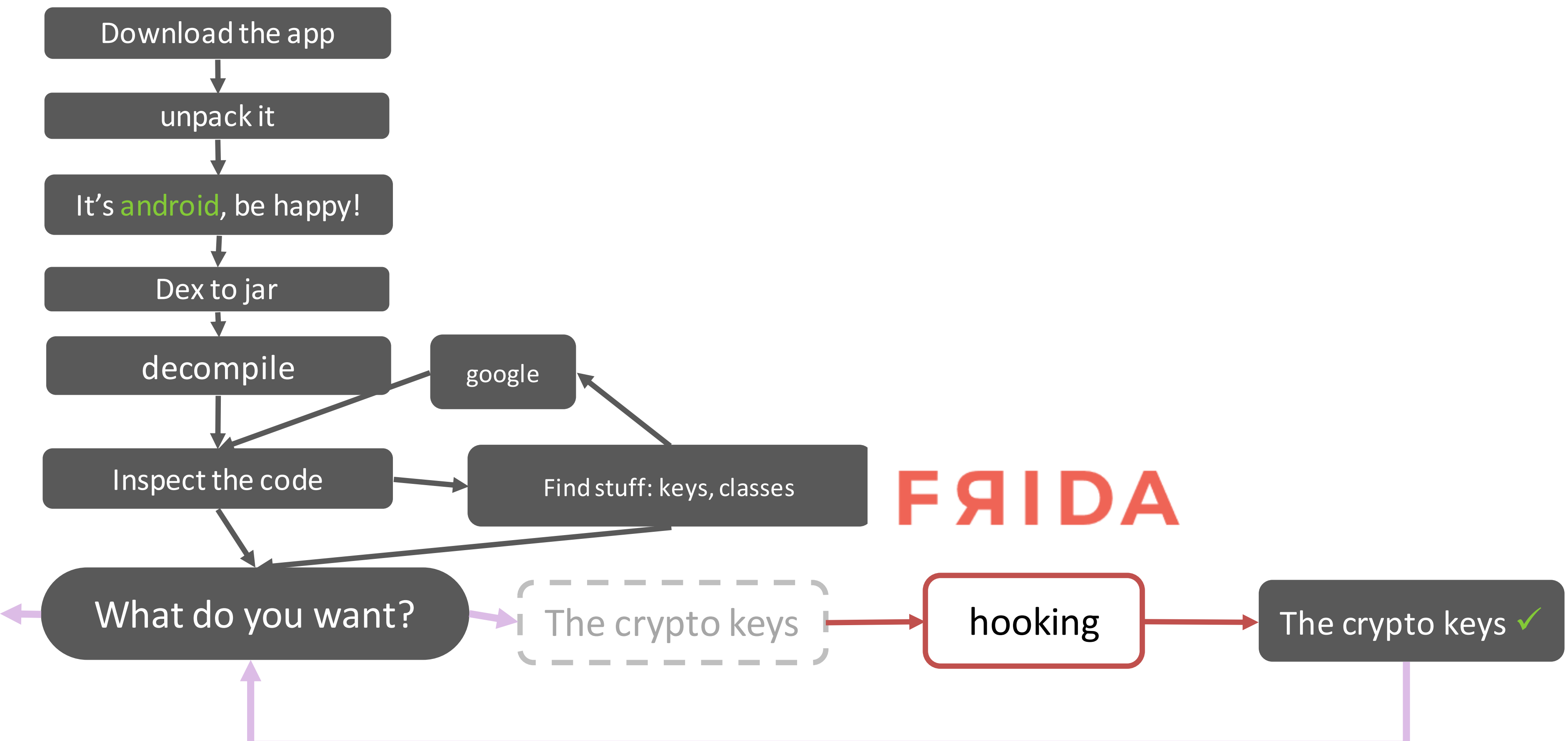
- Key material never enters the application process. When an application performs cryptographic operations using an Android Keystore key, behind the scenes plaintext, ciphertext, and messages to be signed or verified are fed to a system process which carries out the cryptographic operations. If the app's process is compromised, the attacker may be able to use the app's keys but will not be able to extract their key material (for example, to be used outside of the Android device).
- Key material may be bound to the secure hardware (e.g., Trusted Execution Environment (TEE), Secure Element (SE)) of the Android device. When this feature is enabled for a key, its key material is never exposed outside of secure hardware. If the Android OS is compromised or an attacker can read the device's internal storage, the attacker may be able to use any app's Android Keystore keys on the Android device, but not extract them from the device. This feature is enabled only if the device's secure hardware supports the particular combination of key algorithm, block modes, padding schemes, and digests with which the key is authorized to be used. To check whether the feature is enabled for a key, obtain a `KeyInfo` for the key and inspect the return value of `KeyInfo.isInsideSecurityHardware()`.

# Demo 2



Download the app
unpack it
It's android, be happy!
Dex to jar
decompile
google
Inspect the code
Find stuff: keys, classes
get smali
Patch smali
Re-package
Re-sign
Re-install
debug
Make the app debuggable
What do you want?
The crypto keys
hooking
The crypto keys ✓

# Demo 2

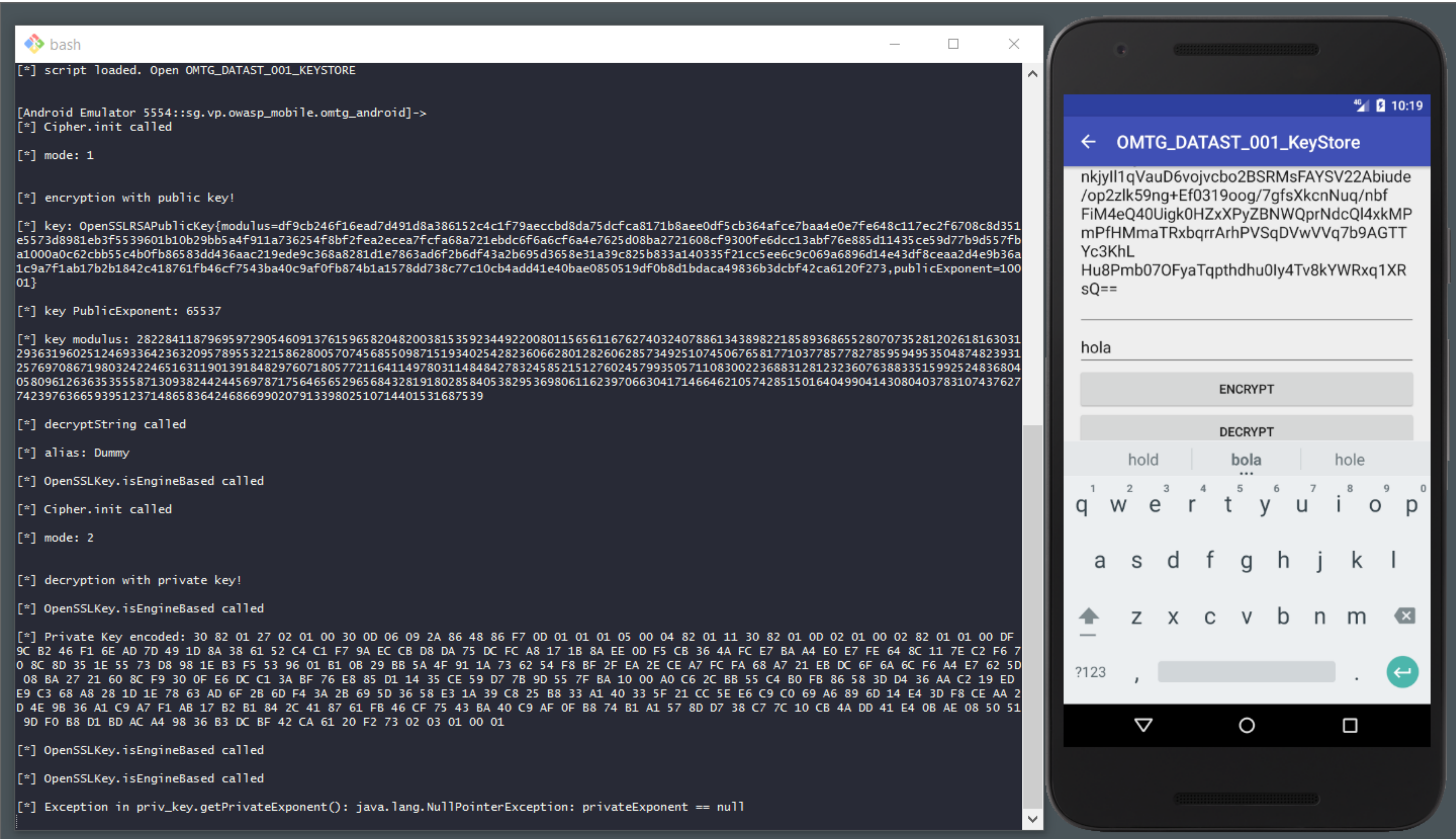


Download the app
unpack it
It's android, be happy!
Dex to jar
decompile
google
Inspect the code
Find stuff: keys, classes
What do you want?
The crypto keys
hooking
The crypto keys ✓

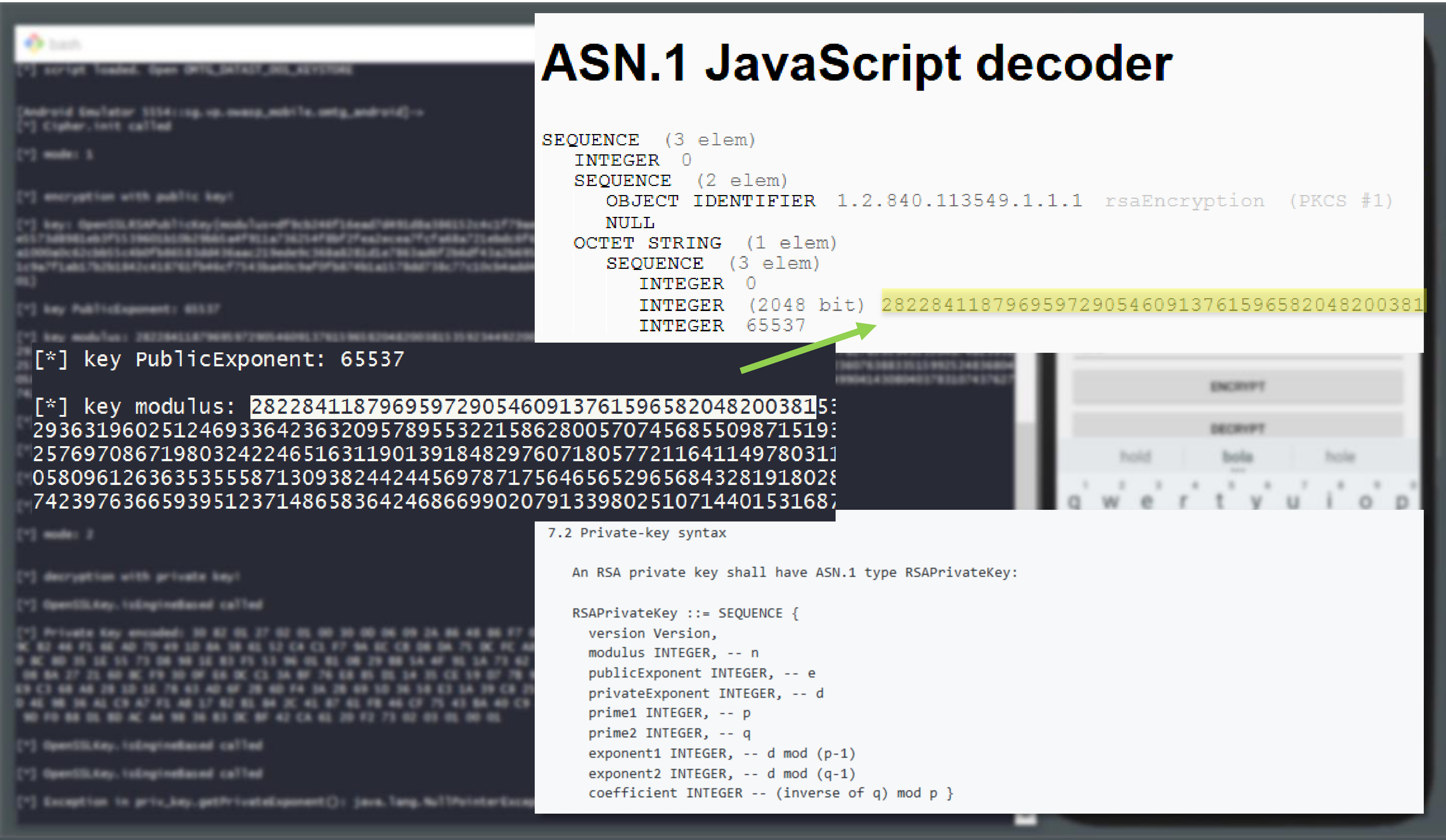
# Demo 2

```
console.log("\n[*] script loaded. Open OMTG_DATAST_001_KEYSTORE\n\n");
var clazz = Java.use("sg.vp.owasp_mobile.OMTG_Android.OMTG_DATAST_001_KeyStore");

clazz.decryptString.overload("java.lang.String").implementation = function (alias) {
  console.log("\n[*] decryptString called");
  console.log("\n[*] alias: " + alias);

  this.decryptString.overload("java.lang.String").call(this, alias);
};

var RSAPublicKey = Java.use("java.security.interfaces.RSAPublicKey");
var RSAKey = Java.use("java.security.interfaces.RSAKey");
var RSAPrivateKey = Java.use("java.security.interfaces.RSAPrivateKey");

var OpenSSLRSAPrivateKey = Java.use("com.android.org.conscrypt.OpenSSLRSAPrivateKey");
var OpenSSLKey = Java.use("com.android.org.conscrypt.OpenSSLKey");

OpenSSLKey.isEngineBased.overload().implementation  = function(){
  console.log("\n[*] OpenSSLKey.isEngineBased called");
  return false;
}

var NativeCrypto = Java.use("com.android.org.conscrypt.NativeCrypto");

var Cipher = Java.use("javax.crypto.Cipher");
Cipher.init.overload('int', 'java.security.Key').implementation  = function(opmode, key){
    console.log("\n[*] Cipher.init called");
    console.log("\n[*] mode: " + opmode);

    if (opmode == 2){
      console.log("\n\n[*] decryption with private key!");
```

# Demo 2


bash
[*] script loaded. Open OMTG_DATAST_001_KEYSTORE
[Android Emulator 5554::sg.vp.owasp_mobile.omtg_android]->
[*] Cipher.init called
[*] mode: 1
[*] encryption with public key!
[*] key: OpenSSLRSAPublicKey{modulus=df9cb246f16ead7d491d8a386152c4c1f79aeccbd8da75dcfca8171b8aee0df5cb364afce7baa4e0e7fe648c117ec2f6708c8d351e5573d8981eb3f5539601b10b29bb5a4f911a736254f8bf2fea2ecea7fcfa68a721ebdc6f6a6cf6a4e7625d08ba2721608cf9300fe6dcc13abf76e885d11435ce59d77b9d557fba1000a0c62cbb55c4b0fb8658 3dd436aac219ede9c368a8281d1e7863ad6f2b6df43a2b695d3658e31a39c825b833a140335f21cc5ee6c9c069a6896d14e43df8ceaa2d4e9b36a1c9a7f1ab17b2b1842c418761fb46cf7543ba40c9af0fb874b1a1578dd738c77c10cb4add41e40bae0850519df0b8d1bdaca49836b3dcbf42ca6120f273,publicExponent=10001}
[*] key PublicExponent: 65537
[*] key modulus: 28228411879695972905460913761596582048200381535923449220080115656116762740324078861343898221858936865528070735281202618163031293631960251246933642363209578955322158628005707456855098715193402542823606628012826062857349251074506765817710377857782785959495350487482393125769708671980324224651631190139184829760718057721164114978031148484278324585215127602457993505711083002236883128123236076388335159925248368040580961263635355587130938244244569787175646565296568432819180285840538295369806116239706630417146646210574285150164049904143080403783107437627742397636659395123714865836424686699020791339802510714401531687539
[*] decryptString called
[*] alias: Dummy
[*] OpenSSLKey.isEngineBased called
[*] Cipher.init called
[*] mode: 2
[*] decryption with private key!
[*] OpenSSLKey.isEngineBased called
[*] Private Key encoded: 30 82 01 27 02 01 00 30 0D 06 09 2A 86 48 86 F7 0D 01 01 01 05 00 04 82 01 11 30 82 01 0D 02 01 00 02 82 01 01 00 DF 9C B2 46 F1 6E AD 7D 49 1D 8A 38 61 52 C4 C1 F7 9A EC CB D8 DA 75 DC FC A8 17 1B 8A EE 0D F5 CB 36 4A FC E7 BA A4 E0 E7 FE 64 8C 11 7E C2 F6 70 8C 8D 35 1E 55 73 D8 98 1E B3 F5 53 96 01 B1 0B 29 BB 5A 4F 91 1A 73 62 54 F8 BF 2F EA 2E CE A7 FC FA 68 A7 21 EB DC 6F 6A 6C F6 A4 E7 62 5D 08 BA 27 21 60 8C F9 30 0F E6 DC C1 3A BF 76 E8 85 D1 14 35 CE 59 D7 7B 9D 55 7F BA 10 00 A0 C6 2C BB 55 C4 B0 FB 86 58 3D D4 36 AA C2 19 ED E9 C3 68 A8 28 1D 1E 78 63 AD 6F 2B 6D F4 3A 2B 69 5D 36 58 E3 1A 39 C8 25 B8 33 A1 40 33 5F 21 CC 5E E6 C9 C0 69 A6 89 6D 14 E4 3D F8 CE AA 2D 4E 9B 36 A1 C9 A7 F1 AB 17 B2 B1 84 2C 41 87 61 FB 46 CF 75 43 BA 40 C9 AF 0F B8 74 B1 A1 57 8D D7 38 C7 7C 10 CB 4A DD 41 E4 0B AE 08 50 51 9D F0 B8 D1 BD AC A4 98 36 B3 DC BF 42 CA 61 20 F2 73 02 03 01 00 01
[*] OpenSSLKey.isEngineBased called
[*] OpenSSLKey.isEngineBased called
[*] Exception in priv_key.getPrivateExponent(): java.lang.NullPointerException: privateExponent == null
10:19
OMTG_DATAST_001_KeyStore
nkjyll1qVauD6vojvcbo2BSRMsFAYSV22Abiude/op2zlk59ng+Ef0319oog/7gfsXkcnNuq/nbfFiM4eQ40Uigk0HZxXPyZBNWQprNdcQl4xkMPmPfHMmaTRxbqrrArhPVSqDVwVVq7b9AGTTYc3KhLHu8Pmb07OFyaTqpthdhu0ly4Tv8kYWRxq1XRsQ==
hola
ENCRYPT
DECRYPT
hold
bola
hole

# Demo 2


ASN.1 JavaScript decoder
SEQUENCE (3 elem)
INTEGER 0
SEQUENCE (2 elem)
OBJECT IDENTIFIER 1.2.840.113549.1.1.1 rsaEncryption (PKCS #1)
NULL
OCTET STRING (1 elem)
SEQUENCE (3 elem)
INTEGER 0
INTEGER (2048 bit) 28228411879695972905460913761596582048200381
INTEGER 65537
[*] key PublicExponent: 65537
[*] key modulus: 282284118796959729054609137615965820482003815
29363196025124693364236320957895532215862800570745685509871519
25769708671980324224651631190139184829760718057721164114978031
05809612636353555871309382442445697871756465652965684328191802
74239763665939512371486583642468669902079133980251071440153168
7.2 Private-key syntax
An RSA private key shall have ASN.1 type RSAPrivateKey:
RSAPrivateKey ::= SEQUENCE {
version Version,
modulus INTEGER, -- n
publicExponent INTEGER, -- e
privateExponent INTEGER, -- d
prime1 INTEGER, -- p
prime2 INTEGER, -- q
exponent1 INTEGER, -- d mod (p-1)
exponent2 INTEGER, -- d mod (q-1)
coefficient INTEGER -- (inverse of q) mod p }

# Takeaways

- ✓ Read the MSTG
- ✓ Use the MASVS
- ✓ Play with Crackmes
- ✓ grepharder
- ✓ Learn FRIDA
- ✓ Learn
- ✓ Contribute!
- ✓ Have fun :)

# References

RTFM STG

# References

- **OWASP Mobile Security Testing Guide**

  https://mobile-security.gitbook.io/mobile-security-testing-guide
  https://github.com/OWASP/owasp-mstg

- **OWASP Mobile Application Security Verification Standard**

  https://mobile-security.gitbook.io/masvs/
  https://github.com/OWASP/owasp-masvs

- **OWASP iGoat - A Learning Tool for iOS App Pentesting and Security**

  https://github.com/OWASP/igoat

- **OWASP MSTG-Hacking-Playground Android App**

  https://github.com/OWASP/MSTG-Hacking-Playground

- **OWASP MSTG Crackmes**

  https://github.com/OWASP/owasp-mstg/tree/master/Crackmes

Thank you, any questions?