

AutoCrypInject v1.0 : Automated Android Asset Security Framework

Author: Sarvesh Aadhitya, Shamalavannan, Karthik

September 20, 2025

Contents

1	Introduction	2
2	Core Features	2
3	System Requirements	2
4	Architecture and Workflow	3
4.1	Architectural Flow	3
4.2	Detailed Workflow	3
5	Core Component Analysis	4
5.1	Cryptography Module	4
5.2	Code Generation and Injection	4
5.3	Smali Patching Engine	4
6	Setup and Usage	5
6.1	Configuration	5
6.2	Execution	5

1 Introduction

AutoCrypInject is a framework designed to increase the security of Android applications by automating internal asset encryption and injecting runtime decryption logic. The tool targets a common vulnerability in mobile applications: the storage of sensitive data, intellectual property, or proprietary libraries as plaintext files within the APK's `assets` directory. By making these assets computationally expensive to access without the application's intended logic, AutoCrypInject raises the barrier against static analysis, asset lifting, and reverse engineering.

2 Core Features

- **Automated APK Decompile & Recompile:** Using Apktool for seamless unpacking and rebuilding of APK files.
 - **Selective Asset Encryption:** Allows the user to interactively select specific assets or all assets for encryption using the **AES-256 (CBC mode)** algorithm.
 - **Dynamic Code Generation:** Generate a custom Java decryption module tailored to the selected assets and embedding the necessary cryptographic keys.
 - **Smali Code Injection:** Converts the Java decryption module into Smali bytecode and intelligently injects it into the decompiled application's source.
 - **Automated Bytecode Patching:** Scans the application's Smali code and automatically patches all invocations of `AssetManager.open()`, `AssetManager.openFd()` to redirect through the injected decryption logic.
 - **Automated Signing:** Signs the final, repackaged APK with a debug key for immediate deployment and testing.
-

3 System Requirements

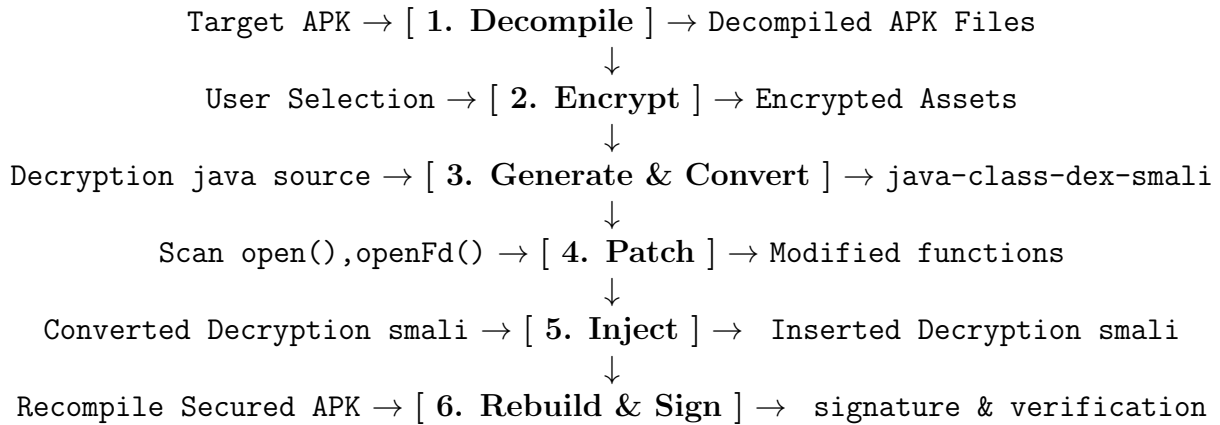
Proper functioning of the toolchain requires the following software:

- **Execution Environment:** Python 3.x
 - **Python Library:** pycryptodome
 - **Android Build/Reverse Tools:**
 - Java Development Kit (JDK) 8 or higher
 - `apktool.jar:decompile apk`
 - `android.jar:Android SDK Components`
 - `d8:Android DEX Compiler`
 - `baksmali.jar:DEX files into Smali code`
 - `uber-apk-signer.jar:APK signing tool supporting signature schemes`
-

4 Architecture and Workflow

The tool operates as a sequential pipeline, where the output of each stage serves as the input for the next. The entire process is orchestrated by the main Python script.

4.1 Architectural Flow



4.2 Detailed Workflow

1. **Initialization:** The script takes a target APK path as input.

2. Decompile

- An APK file is like a ZIP file that contains everything an Android application needs to operate: DEX file, manifest file, resources, assets, native library, etc.
- APK is disassembled into a folder structure containing Smali code, resources, and assets using **apktool**.

3. Asset Encryption Processing:

- The tool lists all files in the **assets/** directory.
- The user selects which assets to encrypt.
- Each selected asset is read, encrypted with AES-256, and overwritten.
- A 16-byte IV is prepended to the ciphertext.

4. Decryption Module Generation:

- A **.java** file is generated containing a class **EncryptedAssetRegistry**.
- This class holds the AES key and a list of encrypted filenames.
- It exposes static methods **open()** and **openFd()** that act as proxies for the standard **AssetManager** calls, handling decryption transparently.

5. **Smali Transformation:** The generated **.java** file undergoes 3 step conversion:

- (a) **javac:** Java source \rightarrow Java bytecode (**.class**)
- (b) **d8:** Java bytecode \rightarrow Dalvik bytecode (**.dex**)
- (c) **baksmali:** Dalvik bytecode \rightarrow Smali source (**.smali**)

6. Injection and Patching:

- The new Smali files are moved into the decompiled APK's Smali directory.
- The script traverses all existing Smali files, searching for specific method call patterns.
- Calls to `AssetManager.open()` and `AssetManager.openFd()` are rewritten to invoke the static methods in `EncryptedAssetRegistry`.

7. Rebuild and Sign

- The modified application structure is recompiled into an APK using **apktool**
 - The resulting package is signed with **Uber** which is signing tool supporting signature schemes v1 (JAR), v2 (APK Signature Scheme), v3, and v4.
 - It simplifies the post-rebuild APK signing and verification process.
 - Finally, A **secured APK** is ready to launch.
-

5 Core Component Analysis

5.1 Cryptography Module

This function implements **AES-256** in **Cipher Block Chaining (CBC)** mode with **PKCS7 padding**. A new, cryptographically secure 16-byte Initialization Vector (IV) is generated for each file to ensure that identical plaintext assets produce unique ciphertexts. The final encrypted file structure is:

$$\text{EncryptedFile} = \text{IV (16 bytes)} || \text{AES-256-CBC(PaddedData)}$$

5.2 Code Generation and Injection

- `generate_encrypted_registry_java()`: Creates the Java source for the decryption logic. The encrypted file list is hardcoded into a `HashSet` for efficient runtime lookups ($O(1)$ average time complexity).
- `convert_java_to_smali_and_inject()`: Orchestrates the multistep conversion from Java source to Smali and places the final files in the target directory.
`decoded_apk/smali/com/decryptassetmanager/`.

5.3 Smali Patching Engine

The patching mechanism is driven by regular expressions and code analysis.

- `find_and_patch_asset_open()`: Using regex to find calls to the `open` method It captures the invocation type (e.g., `invoke-virtual`) and the registers, then rewrites the line to an `invoke-static` call to our injected method.
- `scan_and_patch_open_fd()`: Patches calls to `openFd`. This is more complex as our wrapper requires a `Context` object. The `find_context_register()` helper function uses heuristics to find the register holding the context:

1. If the class is a known Context subclass (e.g., `Activity`, `Service`), it assumes `p0` (the `this` reference).
 2. It checks the method's parameters for a `Context` type.
 3. It performs a reverse search from the call site for an `invoke-virtual ... ->getAssets()` call to identify the object from which the `AssetManager` was retrieved.
-

6 Setup and Usage

6.1 Configuration

Before first use, ensure the paths to the required tools are correctly set as global constants at the top of the script:

```
1 APKTOOL_PATH = 'apktool.jar'
2 AES_KEY = bytes.fromhex("...") # Change for production use
3 # Paths for d8, baksmali, and uber-apk-signer may also need adjustment
```

Security Warning: The default `AES_KEY` is a placeholder. For any real-world application, this key **must** be replaced with a unique, securely generated key. Note that storing the key within the APK itself is a form of obfuscation, not true security.

6.2 Execution

usage:

```
python main.py /path/to/my_app.apk
```

1. Run the script from your terminal, passing the path to the target APK file as the sole command-line argument.
2. The script will display the available assets and prompt for input. You can provide a comma-separated(,) list of filenames to encrypt every asset or type **all** to encrypt all assets.
3. Patching Smali files to decrypt assets at runtime transparently
4. Creating custom class using Java and converting java → Class → Dex → Smali
5. Rebuilding the APK and Signing with Uber for verification.
6. Finally, the Secured APK is ready to Launch.

```
[+] Decoding APK: updated-3.apk
I: Using Apktool 2.11.1 on updated-3.apk with 8 threads
I: Baksmaling classes.dex...
I: Loading resource table...
I: Baksmaling classes2.dex...
I: Baksmaling classes3.dex...
I: Decoding file-resources...
I: Loading resource table from file: C:\Users\karth\AppData\Local\apktool\framework\1.apk
I: Decoding values */* XMLs...
I: Decoding AndroidManifest.xml with resources...
I: Regular manifest package...
I: Copying original files...
I: Copying assets...
I: Copying unknown files...
[+] APK decoded to: decoded_apk
[*] Available assets:
- data.json
- example1.txt
- example2.txt
- example3.txt
- example4.txt
- example5.txt
- example6.txt
- example8.txt
- example9.txt
- image.png
- photo.jpg
- sample.mp4
- sound.mp3
[?] Enter asset filenames to encrypt (comma-separated):
```

Figure 1: Decompilation

```
[?] Enter asset filenames to encrypt (comma-separated): example1.txt,image.png,sound.mp3
[+] 3 valid asset(s) selected for encryption.
b'\xca\xa6\x9b\xaa'\x19k\x02\xed\x06g\x08'kX;'
[+] Encrypted: decoded_apk\assets\example1.txt
b''\xa3\xdf'\xc2C\x04\xf7\x99\x0e\x035\xcc\x129"
[+] Encrypted: decoded_apk\assets\image.png
b'Pr4\x07~w\x02m\x11\x9a\xda?'\xcde\x0f'
[+] Encrypted: decoded_apk\assets\sound.mp3
```

Figure 2: User Selection

```
[?] Enter asset filenames to encrypt (comma-separated): all
[+] All 13 assets selected for encryption.
b'\xb4\x94wX\x0f\xdc\x9f\xee=\xbc\xa9d\x14E$'
[+] Encrypted: decoded_apk\assets\data.json
b'P\x12\xbeC\xea\x9f\xea\x0f\x8c\x07\x9d"E\xa6e;'
[+] Encrypted: decoded_apk\assets\example1.txt
b'\/\xd1\x85\xde\x19b\x062\xd9\x0fd\xccw=;\x90'
[+] Encrypted: decoded_apk\assets\example2.txt
b'\x95\xf6\xfe\xaa(\x029\x00\xbf\x93\xa71\xef\xdfb\xc9'
[+] Encrypted: decoded_apk\assets\example3.txt
b'~\x00\x18\x99\xaf\xa8\xdc\xa1\x08vp\x1b\x951\xcd\xb8'
[+] Encrypted: decoded_apk\assets\example4.txt
b'\x94\x94g\x0e\x1c\x02\x82j\xcb")\xd01\xcc\x89\xc6'
[+] Encrypted: decoded_apk\assets\example5.txt
b'\xa4D95C\x08\x05\x977\xa4\xb3\x80-\xba\xb8\xce'
[+] Encrypted: decoded_apk\assets\example6.txt
b'\xff\x1dn:\x8c\x1b;\xae\xd8\xab\xe4B\xc0^\x93\xcc'
[+] Encrypted: decoded_apk\assets\example8.txt
b'\{ \xa3z\x07'\xb8\xb0\x05\x1b\xf7\x91\x87A\xa8\xfcv'
[+] Encrypted: decoded_apk\assets\example9.txt
b'\xc4\xdd\xbd-j\x02\x086\xccT\x8e\xefQ^\xa9'
[+] Encrypted: decoded_apk\assets\image.png
b'\xc7\x9f\x10\x1d5\x09\x87\x8fA\xe6\x1c/\xaa\xc9Z\x99'
[+] Encrypted: decoded_apk\assets\photo.jpg
b"\x1eX3n\xad Gr\x0f\x7f'\x99\xb5U\x14*"
[+] Encrypted: decoded_apk\assets\sample.mp4
b'\xf1\xbe\xe4\xd3\n\x05~\x8f7\x9d3\xefeol\xc8'
[+] Encrypted: decoded_apk\assets\sound.mp3
```

Figure 3: if all files needs to Encrypt

```
[FOUND] In file: decoded_apk\smali\classes3\com\example\all_test_case\AssetAccessExamples.smali
line 41: invoke-virtual {p1, p2}, Landroid/content/res/AssetManager; >open(Ljava/lang/String;)Ljava/io/InputStream;
[FOUND] In file: decoded_apk\smali\classes3\com\example\all_test_case\AssetAccessExamples.smali
line 277: invoke-virtual {v1, v2}, Landroid/content/res/AssetManager; >open(Ljava/lang/String;)Ljava/io/InputStream;
[FOUND] In file: decoded_apk\smali\classes3\com\example\all_test_case\AssetAccessExamples.smali
line 529: invoke-virtual {v1, v2}, Landroid/content/res/AssetManager; >open(Ljava/lang/String;)Ljava/io/InputStream;
[+] Patched decoded_apk\smali\classes3\com\example\all_test_case\AssetAccessExamples.smali

[FOUND] In file: decoded_apk\smali\classes3\com\example\all_test_case\AssetReader.smali
line 44: invoke-virtual {v2, p1}, Landroid/content/res/AssetManager; >open(Ljava/lang/String;)Ljava/io/InputStream;
[+] Patched decoded_apk\smali\classes3\com\example\all_test_case\AssetReader.smali

[FOUND] In file: decoded_apk\smali\classes3\com\example\all_test_case\AssetUtil.smali
line 37: invoke-virtual {v2, p0}, Landroid/content/res/AssetManager; >open(Ljava/lang/String;)Ljava/io/InputStream;
[+] Patched decoded_apk\smali\classes3\com\example\all_test_case\AssetUtil.smali

[FOUND] In file: decoded_apk\smali\classes3\com\example\all_test_case\CustomAssetManager.smali
line 56: invoke-virtual {v2, p0}, Landroid/content/res/AssetManager; >open(Ljava/lang/String;)Ljava/io/InputStream;
[+] Patched decoded_apk\smali\classes3\com\example\all_test_case\CustomAssetManager.smali

[FOUND] In file: decoded_apk\smali\classes3\com\example\all_test_case\MainActivity.smali
line 69: invoke-virtual {v1, p1}, Landroid/content/res/AssetManager; >open(Ljava/lang/String;)Ljava/io/InputStream;
[FOUND] In file: decoded_apk\smali\classes3\com\example\all_test_case\MainActivity.smali
line 652: invoke-virtual {v3, v4}, Landroid/content/res/AssetManager; >open(Ljava/lang/String;)Ljava/io/InputStream;
[FOUND] In file: decoded_apk\smali\classes3\com\example\all_test_case\MainActivity.smali
line 765: invoke-virtual {v8, v9}, Landroid/content/res/AssetManager; >open(Ljava/lang/String;)Ljava/io/InputStream;
[+] Patched decoded_apk\smali\classes3\com\example\all_test_case\MainActivity.smali
```

Figure 4: Patched open function

```
[FOUND] In file: decoded_apk\smali\classes3\com\example\all_test_case\MainActivity.smali
line 164: invoke-virtual {v8, v1}, Landroid/content/res/AssetManager; >openFd(Ljava/lang/String;)Landroid/content/res/AssetFileDescriptor;
[FOUND] In file: decoded_apk\smali\classes3\com\example\all_test_case\MainActivity.smali
line 735: invoke-virtual {v8, v6}, Landroid/content/res/AssetManager; >openFd(Ljava/lang/String;)Landroid/content/res/AssetFileDescriptor;
[FOUND] In file: decoded_apk\smali\classes3\com\example\all_test_case\MainActivity.smali
line 791: invoke-virtual {v10, v11}, Landroid/content/res/AssetManager; >openFd(Ljava/lang/String;)Landroid/content/res/AssetFileDescriptor;
[+] Patched openFd decoded_apk\smali\classes3\com\example\all_test_case\MainActivity.smali
```

Figure 5: Patched openFd function

```
[*] Generated Java class with decryption: temp_java\EncryptedAssetRegistry.java
[*] Compiling Java file...
[*] Converting .class to .dex using d8...
[*] Disassembling .dex to .smali...
[*] Injected smali files into: decoded_apk\smali\com
```

Figure 6: converted java to smali

```
[*] Recompiling APK...
I: Using Apktool 2.11.1 on recomplied_app.apk with 8 threads
I: Checking whether sources have changed...
I: Checking whether sources have changed...
I: Smaling smali folder into classes.dex...
I: Checking whether sources have changed...
I: Smaling smali_classes2 folder into classes2.dex...
I: Smaling smali_classes3 folder into classes3.dex...
I: Checking whether resources have changed...
I: Building resources with aapt2...
I: Building apk file...
I: Importing assets...
I: Importing unknown files...
I: Built apk into: Output\recompiled_app.apk
[+] Recompiled APK saved at: Output\recompiled_app.apk
```

Figure 7: Recompiling the APK

```
[*] Signing APK using User APK Signer...
source:
  D:\Wall_shared\ProtectAI\Module-B\Specific\largesize+timesall-opt\Output
binary: lib/windows-32_0_2\libwinpthread-1.dll
C:\Users\Karth\AppData\Local\Temp\uapksigner-5249308636048735085
zipalign location: BUILD_IN
C:\Users\Karth\AppData\Local\Temp\uapksigner-5249308636048735085\win-zipalign_32_0_2.exe8432771008780152234.tmp
keystore:
  [0] 161a0018 C:\Users\Karth\AppData\Local\Temp\temp_3873620906306742773_debug.keystore (DEBUG_EMBEDDED)
01. recomplied_app.apk

SIGN
file: D:\Wall_shared\ProtectAI\Module-B\Specific\largesize+timesall-opt\Output\recompiled_app.apk (103.21 MiB)
checksum: 80631905efcd90d245b22055d4662992e9abd72cc4eca0950cb9d7675923e447 (sha256)
- zipalign success
- sign success

VERIFY
file: D:\Wall_shared\ProtectAI\Module-B\Specific\largesize+timesall-opt\Output\recompiled_app-aligned-debugSigned.apk (103.29 MiB)
checksum: a9891ac28a543663f3a0f00e719248e2654c0bfae78971d69cc08006262cb (sha256)
- zipalign verified
- signature verified [v2, v3]
  Subject: CN=Android Debug, OU=Android, O=US, L=US, ST=US, C=US
  SHA256: 1e080903ae79a3272151064e73440d130894e0954161b62544ea8f187b5953 / SHA256withRSA
  Expires: Fri Mar 11 01:40:05 IST 2044

[Wed Jul 30 16:07:10 IST 2025][v1.3.0]
Successfully processed 1 APKs and 0 errors in 4.03 seconds.
[+] APK signed successfully.
```

Figure 8: Signing the APK