ECTester

Quick Start Guide

Dr. Ján Jan#ár

ECTester: Quick Start Guide

by Dr. Ján Jančár

Publication date 1 August 2019

Table of Contents

1. Introduction 1
    - About this Manual 1
    - Architectures 1
2. ECTester Setup 2
    - Requirements 2
    - ECTester Setup 2
3. Java SmartCards 3
    - Section title 3
4. Java Libraries 4
    - Section title 4
5. Native Libraries 5
    - Build environment 5
    - Build library shims 5
    - Build ECTesterStandalone.jar 6
    - Run the collection 7
    - Plot the graphs 7
6. Heat Maps 8
    - Introduction 8
    - Good map 8
    - Bad map 9

A. CVE Findings 11
    - Java SmartCards 11
    - Java Libraries 11
    - Native Libraries 11

Index 12

Chapter 1. Introduction

ECTester is a Java based elliptic curve cryptography security testing and evaluation tool (ST&E) written by Ján Jančár. The tool tests behavior of elliptic curve cryptography implementations on JavaCards (`TYPE_EC_FP` and `TYPE_EC_F2M`) and selected software libraries.

ECTester has successfully tested a number of SmartCards and libraries, including XXX, YYY, ZZZ, Botan, Crypto++ and OpenSSL.

ECTester also produces heat maps to display the leaks as a dot density graph.

About this Manual

The ECTester book has five chapters. The first chapter details ECTester setup. The chapter discusses how to setup a rig to test Java and native libraries. The second chapter is Java SmartCards. The chapter discusses how to test Java SmartCards. The third chapter is Java libraries. The chapter discusses how to test Java libraries. The fourth chapter is Native libraries. The chapter discusses how to test native libraries like Botan, Crypto++ and OpenSSL. The final chapter is an appendix. The appendix provides CVE findings discovered in Java SmartCards, Java Libraries, and Native Libraries.

Architectures

The examples in the book use i686 and x86_64 for testing. ECTester will work equally well on other architectures, like ARM, Aarch64 and PowerPC.

Chapter 2. ECTester Setup

This chapter discusses system requirements and ECTester setup on Linux. Windows Setup will be similar to Linux.

Requirements

ECTester is a Java based application. Java 8 is required to build and run the program. You should install the Java 8 JDK to install the Java compiler and other JDK tools. On Debian and derivatives install package `openjdk-8-jdk`. On Red Hat and derivatives, like Fedora, install `java-1.8.0-openjdk` and `java-1.8.0-openjdk-devel` packages.

In addition to the JDK you should install `ant` package.

Java SmartCard testing requires XXX.

Java Libraries testing requires XXX.

Native Libraries testing requires common build tools, like cmake, make, a C or C++ compiler, an assembler, and a linker. You may also need Autotools like autoconf and automake, depending on the library.

On Debian and derivatives you can install the `build-essentials` package. On Red Hat systems, like CentOS and Fedora, you should install the `gcc`, `gcc-c++`, `make`, `cmake`, `autoconf` and `automake` packages.

ECTester Setup

The source code for ECTester is located at crocs-muni GitHub. You can clone it with the following command.

```
git clone https://github.com/crocs-muni/ECTester
```

Chapter 3. Java SmartCards

TODO: discuss testing Java SmartCards.

Section title

TODO: discuss the details of testing Java SmartCards.

Chapter 4. Java Libraries

TODO: discuss testing Java libraries.

Section title

TODO: discuss the details of testing Java libraries.

Chapter 5. Native Libraries

Many popular elliptic curve libraries are written in C, C++ and other languages. This chapter discusses how to test native libraries like BoringSSL, Botan, Crypto++ and OpenSSL.

Each native library you want to test will need a shim library for use in the ECTester program. The shim library will depend on the native library, and have a name like `boringssl_provider.so`, `botan_provider.so`, `cryptopp_provider.so` and `openssl_provider.so`.

There are fours steps to test a native library using ECTester. The first step is build the shim library. The second step is build `ECTesterStandalone.jar`. The third step is running the collection to collect data. The fourth step is plot the graphs to visualize the data.

The steps below use Crypto++ as an example. You should use the appropriate library name when testing other libraries, such as BoringSSL, Botan, or OpenSSL.

Build environment

There are two important environmental variables that should be set in your environment. First, you should set `JAVA_HOME`. The tooling uses `JAVA_HOME` to locate native Java library headers, like `jni.h`. Second, you should set `PKG_CONFIG_PATH`. The makefile uses `PKG_CONFIG_PATH` to locate a library's `*.pc` file.

Linux distributions uses different paths for libraries. `PKG_CONFIG_PATH` attempts to abstract away the path differences but it needs help at times. On Debian and derivatives, like Ubuntu, the library path is `/lib` and `/usr/lib`. On Red Hat and derivatives, like Fedora, the library path is `/lib64` and `/usr/lib64` on 64-bit machines. You may need to tweak the makefile and other configuration files for distros like Fedora.

Build library shims

There are two ways to build the shared libraries. First, from the ECTester and `jni/` directory, build the shim using the `Makefile`.

```
cd ECTester
cd src/cz/crcs/ectester/standalone/libs/jni
PKG_CONFIG_PATH=/usr/lib64/pkgconfig/ make cryptopp
```

Note the use of `make cryptopp` to build just the Crypto++ library. If all goes well then you should see output similar to the following.

```
g++ -I/usr/local/include  -fPIC -I"/usr/lib/jvm/java-1.8.0-o
penjdk-1.8.0.212.b04-0.fc30.x86_64/include" -I"/usr/lib/jvm/
java-1.8.0-openjdk-1.8.0.212.b04-0.fc30.x86_64/include/linux
" -I. -O2 -c cryptopp.cpp
g++ -fPIC -I"/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.212.b04-0
.fc30.x86_64/include" -I"/usr/lib/jvm/java-1.8.0-openjdk-1.8
```

```
.0.212.b04-0.fc30.x86_64/include/linux" -I. -O2 -c cpp_utils
.cpp
cc -o lib_timing.so -shared -fPIC -I"/usr/lib/jvm/java-1.8.0
-openjdk-1.8.0.212.b04-0.fc30.x86_64/include" -I"/usr/lib/jv
m/java-1.8.0-openjdk-1.8.0.212.b04-0.fc30.x86_64/include/lin
ux" -I. -O2 -Wl,-soname,lib_timing.so c_timing.c
g++ -fPIC -shared -O2 -o cryptopp_provider.so -Wl,-rpath,/li
b cryptopp.o cpp_utils.o -L. -L/usr/local/lib -lcryptopp  -l
:lib_timing.so
```

Tip

Crypto++ is unique among packages because it has different names depending on the distro. On Debian and derivatives, like Ubuntu, the library name is `libcrypto++`, and the development package with header files is `libcrypto++-dev`. On Red Hat and derivatives, like Fedora, the library name is `cryptopp`, and the development package with header files is `cryptopp-devel`.

If you need to change the library name then use `sed` on the file. For example, to change the library name in the makefile, run the command

```
sed -i 's/libcrypto++/libcryptopp/g' Makefile
```

Second, from the ECTester source code directory, use `ant` to build the source code.

```
cd ECTester
ant -f build-standalone.xml libs
```

This will compile the shared libraries which ECTester uses via the Java Native Interface to work with native libraries. That makefile uses `pkg-config` to find Crypto++, so all of that should apply.

Build ECTesterStandalone.jar

Build `ECTesterStandalone.jar` with `ant -f build-standalone.xml jar`, possibly run this twice. If all goes well then the build should finish with the message *BUILD SUCCESSFUL*.

```
Buildfile: /home/test/ECTester/build-standalone.xml
-pre-init:
-init-private:
-init-user:
...

-do-jar-copylibs:
 [copylibs] Copy libraries to /home/test/ECTester/dist/lib.
 [copylibs] Building jar: /home/test/ECTester/dist/ECTesterS
tandalone.jar
     [echo] To run this application from the command line wi
thout Ant, try:
```

```
     [echo] java -jar "/home/test/ECTester/dist/ECTesterStan
dalone.jar"
...

jar:

BUILD SUCCESSFUL
Total time: 11 seconds
```

Run the collection

Go to `dist/` and run the collection.

```
cd dist/
java -jar ECTesterStandalone.jar list-libs
```

It should list Crypto++ as available, together with other libraries that ECTester was able to compile the shims for (they were available in the system) and with pure Java libraries. If it doesn't list Crypto++, something went wrong with the above steps.

Then do something like the following.

```
java -jar ECTesterStandalone.jar ecdsa \
  -n 500000 -nc secg/sect233r1 \
  -o out.csv Crypto++
```

That should perform 500k signatures over the sect233r1 curve and output the data into out.csv.

Plot the graphs

Go to `util/`, run the `plot_dsa.ipynb` Jupyter notebook and follow the instructions there to plot the ECDSA data from out.csv.

Chapter 6. Heat Maps

This chapter discusses interpreting the heat maps produced by ECTester.

Introduction

Interpreting the heat maps produced by ECTester is one of the keys to understanding and fixing timing leaks in programs and libraries.

Good map

The image below shows an ideal heat map.

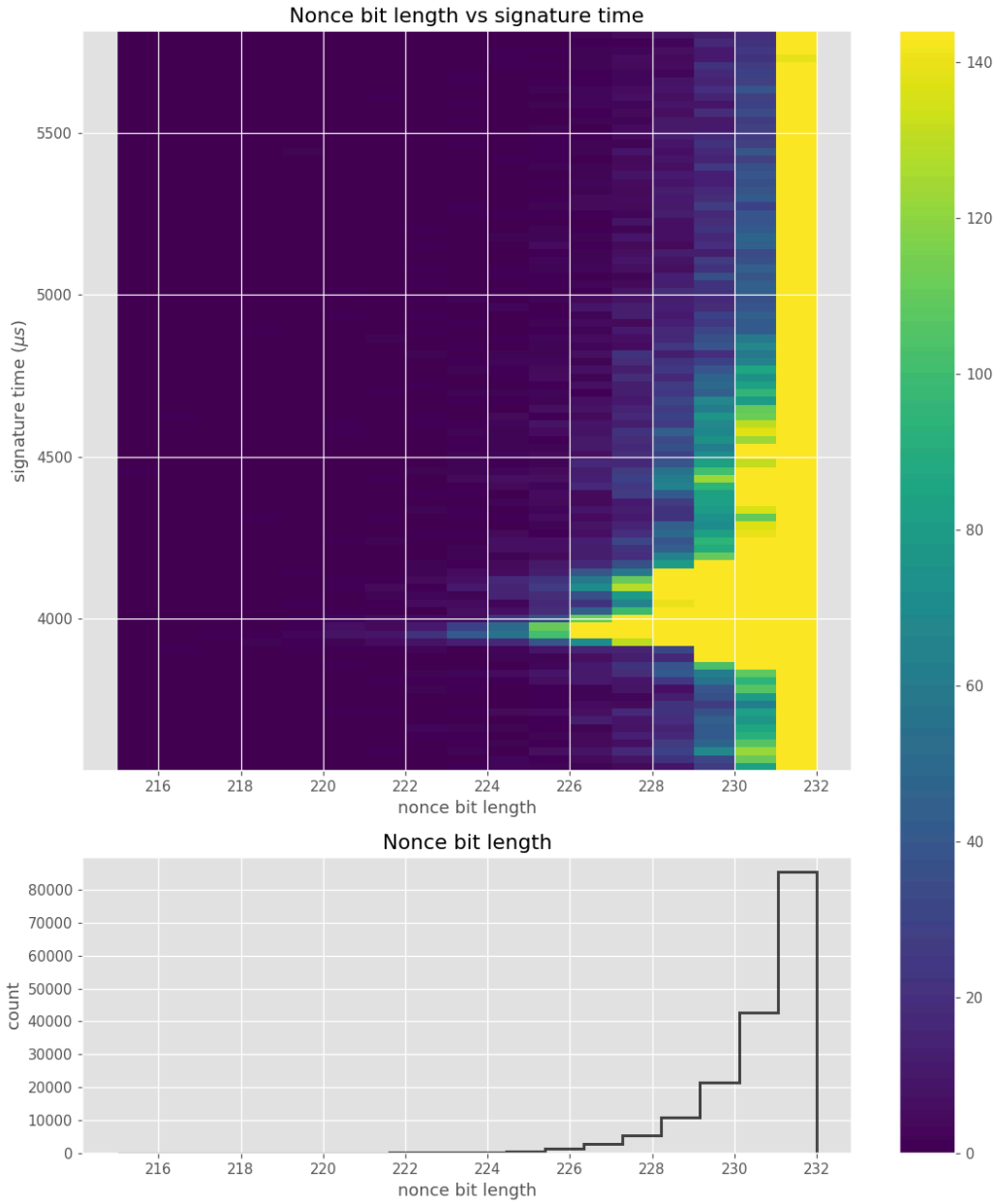

Image of a good heat map. The algorithm has little to no leakage.

TODO: explain why this is a good heat map.

Bad map

The image below shows an ideal heat map.

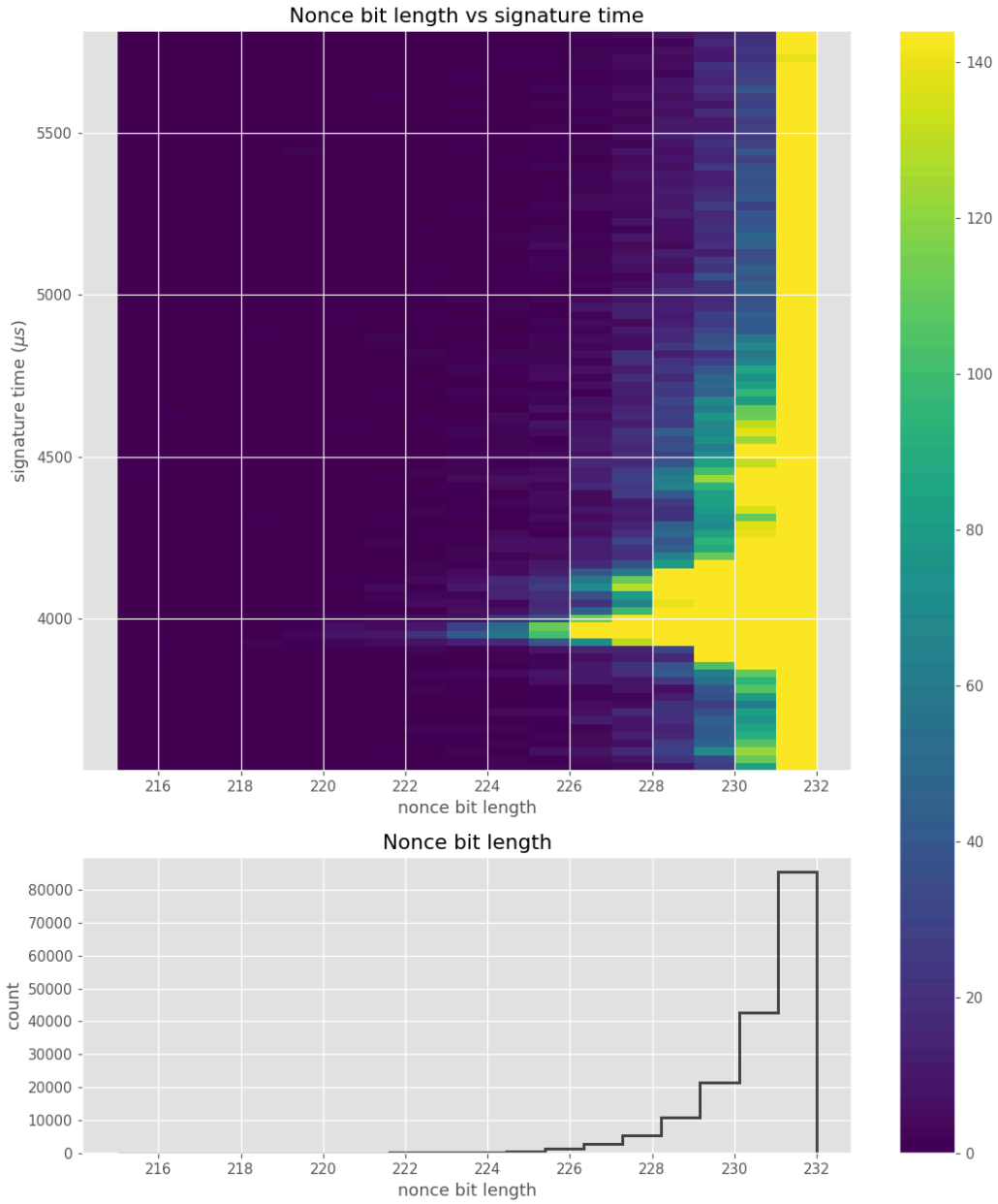

Image of a bad heat map. The algorithm is leaking information.

TODO: explain why this is a bad heat map.

Appendix A. CVE Findings

The tables below presents Common Vulnerabilities and Exposures (CVE) found by ECTester.

Java SmartCards

The table below shows vulnerabilities discovered in Java SmartCards.

SmartCard	CVE	Comment
Unknown	CVE-2019-NNNN	Unknown

Java Libraries

The table below shows vulnerabilities discovered in Java libraries.

Library	CVE	Comment
BouncyCastle	CVE-2019-NNNN	Unknown
Sun EC	CVE-2019-NNNN	Unknown

Native Libraries

The table below shows vulnerabilities discovered in native libraries.

Library	CVE	Comment
BoringSSL	CVE-2019-NNNN	Unknown
Botan	CVE-2019-NNNN	Unknown
Crypto++	CVE-2019-14318	Information leaks in prime fields and binary curves.
Intel PPC	CVE-2019-NNNN	Unknown
libgcrypt	CVE-2019-NNNN	Unknown
libtomcrypt	CVE-2019-NNNN	Unknown
MatrixSSL	CVE-2019-NNNN	Unknown
MbedTLS	CVE-2019-NNNN	Unknown
Microsoft CNG	CVE-2019-NNNN	Unknown
OpenSSL	CVE-2019-NNNN	Unknown
wolfSSL	CVE-2019-NNNN	Unknown

Index