TECHNISCHE
UNIVERSITÄT
DRESDEN

Faculty of Computer Sciences Institute of Systems Architecture, Chair of privacy and security

Bachelor thesis

Performance evaluation of cryptographic algorithms on Android devices

Juan Ignacio Pérez de Algaba Sierra

16th September 2018

Supervisor

Dr.-Ing. Stefan Köpsell

Contents

Abstract

In this document we will research how the performance of different cryptographic libraries (mbedTLS, OpenSSL, BoringSSL, wolfCrypt and Bouncy Castle) varies on different Android devices in order to help developers be able to find a suitable solution for every possible situation. To test the performance, we will perform different benchmarks using different algorithms, such as AES-CBC, RSA, Diffie Hellman or MD-5. To perform these tests we have developed our own app that uses all these libraries and algorithms to help us perform the different benchmarks. We will first prove that using C and C++ Libraries on Java app does not affect the performance of the native libraries. Then we will perform the cryptographic operations on the different devices to see which library provides a better performance. In the majority of the cases OpenSSL will be the best library to use, but there are cases where developers will need to apply better (or more efficient) solutions. We will later perform different tests at different times of the day to see if there is any difference in the results and surprisingly in some cases, the performance of an algorithm varies significantly. Ultimately we will also see how the temperature of the CPU of the different devices varies during the processes to see if these operations could harm the battery life of the phones and unexpectedly, we will show that high-performance devices have a worse performance than low-spec devices.

1 Introduction

After the Facebook-Cambridge Analytica data scandal the world is shocked [Blo18]. People realized that their information was worth a lot of money and companies were using their data not only to do our browsing experience better but also to make business with them [McR18]. They sell our precious information to other companies to offer personalized ads or to influence politics. In addition to this, we are heading to a more connected world where we use our mobile phone for everything, not only calling our friends but also to check our bank account, share photos with our friends and write our shopping list. We buy a new mobile phone and thanks to the cloud our old photos, contacts and data are already there. Our whole life is online and the majority of the time, we don't even realize that.

Due to these two linked facts, people are getting more worried about the data they share on the Internet and give to online companies [Mas16].

Also, users do not only want to protect themselves against evil companies, but also they have to protect their mobile phone against the typical and old threats: according to some studies, 97% of the mobile malware is on Android [Kel14]. As this operating system is the most used on the planet [17], it is understandable that the goal of the majority of viruses, dangerous software and attackers, aimed accordingly mostly at Android, is to steal user's information, such as their private conversations and their login credentials for the web services. Normally users do not really know how cryptography works, they just want a way that allows them to protect their data and here it is where developers have to play an important role.

Developers have listened to the voice of the people and they have released apps (i.e. Telegram and Signal) that allow to protect communications, send encrypted messages,etc. Althouth this is a good initiative and an alternative to the standard and known applications, sometimes the way these cryptographic operations are implemented becomes, in the end, a problem instead of a solution. Every company wants to keep their released products secret, and in order to do so, they will not disclose to the user how their communications were protected.

After having identified one of the main problems of this technological era, we should agree on the fact that users should have the choice to be able to defend themselves against these attackers protecting their information in a safe way and developers should be able to know how they should protect the user's data in the most effective way taking into account every different device and without harming the useful life of the mobile phone.

To help developers perform this task, we have developed an Android app that performs different cryptographic operations using various algorithms from very different libraries to check the performance on a collection of devices, taking into account the speed of each operation and how the temperature of the mobile phone changes during the process. This app will help developers decide which library and algorithm they should implement in their apps according to every phone or Android version. In the following pages, we explain how this app works and which are the results that we have obtained after performing the benchmarks on different devices.

2 Benchmarking

2.1 What is benchmarking?

According to Philip J. Fleming and John J. Wallace, **benchmarking** is the act of running a computer program, a set of programs of other operations, in order to assess the relative performance of an object, normally by running a number of standard tests and trials against it [FW86]. Although this might sound like a very fixed process, the set of programs that are run have been designed according to the judgment of a developer. This means that every benchmark is subjective and there is not an absolute truth, that means that different benchmarking apps will get different results. Because of this fact, first we had to decide which cryptographic program suits better our needs. We were interested in performing benchmark on different cryptographic algorithms and as we did not find any benchmarking app that could help us with our task, we decided to implement our own solution.

2.2 Methodology

Before explaining how our tests work, we should first explain which is the methodology normally used when benchmarking cryptographic algorithms. There are libraries, such as Crypto++ that already have their own benchmarking suite that allows an easy comparison with other libraries [unk]. These benchmarks measure 2 different things. First of all, the MiB/Second and the cycles per byte. To do this, the suite will assign a time during which the library will be executing the algorithm and after the time is finished, it will return the results. This is usually the normal way of measuring the different libraries, but this can lead to some problems. First of all, after every cryptographic operation, the program will measure the time that it was needed to perform the operation. This is not a problem if we are working with large amount of data, but on the contrary, if we are working with small-sized data, it could be that the measuring operation will take more time than the cryptographic operation and we will be getting adulterated results. Secondly, all these benchmarks are really closed and the user can not really test different parameters. For example, in Crypto++ if they want to test AES-CBC with a 128 bytes key, the user can not choose the block size he wants to encrypt and the benchmark will try all possible block sizes automatically. Although this is a good idea for standardizing the tests and getting general results that can be compared with other results, it takes away the freedom to choose specific values. Another general problem of these benchmarking suites is that every different developers takes a different approach to them, so at the end, the results can not be compared.

Although there have been some efforts to standardize the different results of different benchmarks, for example, FELICS(Fair Evaluation of Lightweight Cryptographic systems)

[Din+15], the problem is that these solutions don't work for every possible library and/or algorithm. Also there are additional aspects that they don't take into account. In mobile devices, the temperature is very important as they are objects that the user will be holding in his hand for a long time, but as nowadays these devices are an alternative for computers, these cryptographic suites do not worry about this factor. Also these frameworks do not allow to store different libraries, this means that if a user wants to analyze the same library on different devices, the user will have to set up everything again because the different configurations for the different libraries and architectures can not been stored.

2.3 Influences over the benchmarks

Although we will be measuring how the different devices behave performing the different devices on the different libraries, we should consider that our benchmarks will not be 100% accurate but we will have very narrow approximations. The main problem of performing the benchmarks is the huge amount of factors that play a role at every moment. The results of the different tests should only be influenced by the CPU (the different architectures, like x86, x64 or ARMv7 and the different providers, like NVIDIA, Qualcomm or Intel) and the RAM of the device. These were the information we were interested in, but at the end these were not the only factors that affect the performance. We also need to have in mind that every device is running at the same time not only the different tests, but also there are a lot of services running at the same time (Internet related services, battery management, etc...), which will the CPU from running at full power . These are software related services but also there are hardware and manufacturing problems. First of all, every mobile phone has a battery. This battery is always located near the CPU (due to the small sizes of the devices) and usually while the battery is working, it will get very high temperatures. The temperature of the CPU can be influenced by the battery and this will result in lower general power of the processor. Also the materials used to build the device are very important, as it makes a difference if the devices are built from plastic or different metals, as they can have a better thermal conductivity or need more time to heat themselves [unk16]. Apart from these hardware/software aspects, it is also important to remind that electronic devices (computer, smart phones, ...) slow down over time due to several reasons, so the results of the exact same benchmark on the device will vary over time [Mer17].

2.4 Related work

Before finishing this chapter it is important to mention how have other researchers approached to benchmarking cryptographic algorithms on Android Devices. Unfortunately, there are not so many related papers. One of the first papers written about this topic is called "Standard Method of Evaluating Cryptographic Capabilities and Efficiency for Devices with the Android Platform" by Michael A. Walker [Wal10]. The author developed an app using only one Java library[1] and tested it on only one device. This app is not open-source so the comparisons with

[1]The javax.crypto package

our own app are impossible. The most significant and related research to our own paper is the one written by David González, Oscar Esparza, Jose L. Muñoz, Juanjo Alins and Jorge Mata called “Evaluation of Cryptographic Capabilities for the Android Platform” [Gon+15]. In this paper, they also develop an app to test the cryptographic capabilities of some libraries. The main difference between this paper and our work is that whereas they only use Java libraries and only one device, we built the applications using also C native libraries and we tested it on 4 different devices. Resulting this in more data and being easier to decide which library has a better performance. Unfortunately, the app that this team has developed is neither open-source and nor can we see how the functions are really built and thus, we can not compare it with our own application.

As we have seen in these different papers, the current trends in developing Android benchmarking applications are that app is not open source and the tests are done by the researchers, so the users can’t have access to the application and they can not modify the values that are tested. As we will see in the next chapters, we will try to tackle this problems applying other solutions.

3 Building the app

In this chapter we would like to explain how the app was built, the libraries that we implemented and which algorithms were used to perform the tests.

3.1 App

As the idea was to perform the tests on Android devices, the app had to be built using Java. To help us with this task we used Android Studio[autb], the Google's IDE that is ready to work on the Android operating system and has several advantages over another IDEs, such as the integrated ADB[1] or the option to work for different Android versions. As we are also using C and C++ libraries, we also used the NDK (Native development Kit) to help us run the native code on Java platforms. The NDK uses CMake [autf] as a compiler and allows the developer to run code locally that is aimed to be executed on different architectures. As the NDK is already integrated in Android Studio, its implementation was simple once its functioning was learned. To develop an Android app, it is also necessary to decide which will be the targeted SDK(*targetSdkVersion*), the compiled SDK version (*compileSdkVersion*) and the minimum version where the app will be able to run (*minSdkVersion*). We chose as the compile and targeted SDK the version 26, as at the time of programming the app, it was the latest API-Level available, which was introduced with Android O(reo). For the minimun SDK version, we decided to use the API-Level 15(Android Ice Cream Sandwich), as according to the Android Studio's advices, the app will be able to run on every Android mobile phone. This seemed the best version as we wanted to perform the tests on as much devices as possible and with different OS versions.

Additionally, we made the app available with "armeabi-v7a" and "86""architectures. There are not any official statistics from Google, but according to Unity[2], 98,1% of the mobile phones use ARMv7, whereas 1,7% of the devices use x86[aut17]. Because of these stats, we considered that it was unnecessary to implement any other architecture. These architectures are used by CMake to compile each library in a different way, depending on the device the app is installed on.

Once the main structure of the app was finished, we had to decide which libraries we wanted to use.

3.2 Libraries

For our libraries' choice, we chose the ones we considered more interesting to use. In this section we will explain what makes every one of these libraries attractive to use.

[1]Android Debug Bridge

[2]A worldwide extended "cross-platform" game engine

Before explaining the libraries, we would like to explain why we didn't use the Google's standard Android Security provider, called Conscrypt. The main problem with this provider is that it is being constantly updated and implemented in Android, this means that for every Android version every algorithm will perform differently or it won't even be implemented. For example, in the version for Android O, the GCM parameter for the AES algorithm was added and for previous versions, this option wasn't available[Goo18].

3.2.1 Bouncy Castle

The first choice was also the most logical option. Since Bouncy Castle is built entirely in Java[aute] , it suits perfectly our needs. Bouncy Castle is an API[3] looked after by an Australian Charity, the Legion of the Bouncy Castle Inc. This organization takes care that the API is still open-source, free and maintained. Although this library was released in the year 2000, it still gets updates and new releases.

The implementation of Bouncy Castle in Android apps is not so good as there are some compatibility problems between Android libraries and this security provider. Because of that, we used Spongy Castle[auto]. This library acts as a wrapper for the original Bouncy Castle and avoids all the compatibility problems. In our App the used version of Bouncy Castle is 1.48.

3.2.2 mbedTLS

Our second library is mbedTLS (or previously PolarSSL)[autg]. This security provider is an implementation of the TLS and SSL protocols with its respective cryptographic algorithms. This is the first of our libraries which is written in C. This library is younger than Bouncy-Castle as it was released on 2006 and it is still being updated and documented. One of the main features of mbedTLS is its minimal coding footprint, requiring only 60kb of program space and under 64 KB of RAM to work. Because of this, it is oriented to be used on embedded systems. Android devices do not belong to this class of systems but we found it very interesting to research how a so versatile library would run on mobile devices.

The compilation of this library was done with CMake and following its GitHub tutorial[auth]. The used version was the latest release at the time of designing the app, 2.8.0.

3.2.3 wolfCrypt

Our next choice for the library was wolfCrypt (CyaSSL or "yet another SSL")[autq]. wolfCrypt is also a library written in C and C++ and is targeted for embedded systems because of its size and speed. This library, like mbedTLS also has a small footprint size and a low runtime memory so trying it was interesting because this would mean that if the results on the different devices were good, developers could implement this library on their apps without sacrificing the storage of the device.

As our previous library, wolfCrypt was also released with wolfSSL, as an implementation of the SSL and TLS protocols in the year 2004.

[3]Applicaton programming interface

To use this library on our app we used the wolfCrypt JNI package that wolfSSL offers on its GitHub repository[autr]. This package provides a Java, JNI-based interface to the native wolfCrypt. It also provides an JNI wrapper, but we have not used it on our project, we used plain C code. For this library we used the version 3.13.0.

3.2.4 OpenSSL

Our next library is OpenSSL[autj]. OpenSSL is almost the standard of cryptographic libraries, as a lot of systems use it for establishing secure communications. It was released in 1998, being the oldest library that we used and like for our last 2 libraries, it has also been written in C. We would expect that being so old, its performance would not be as good as in the more modern ones, but maybe it has had more time to be optimized. As every library that we have used in the project, this is also open-source.

To compile OpenSSL to run on Android architectures we followed the tutorial that they propose on their website[autk]. This time, we used version 1.1.0f.

3.2.5 BoringSSL

Our last library is BoringSSL[autc]. It may seem that testing BoringSSL on the different devices is just a waste of time, because this library is a fork of OpenSSL. BoringSSL is, however, the Google's fork of OpenSSL, meaning that they have adapted this library to work better for their own needs, and as Android is an important part of Google, it has also been modified to work on this operating system. Currently, it is being used as the SSL Library on Android (but it's not part of the NDK). As in mbedTLS, Google recommends compiling the library on CMake to make it compatible with Android devices[autd]. BoringSSL does not use a version system, Google just keeps updating the library and the only version control system that we can obtain is the last commit that they did of it. For the version that we used, it was the commit "fe7a174" based on the 1.1.0 version of OpenSSL, as stated in the file `base.h`.

It is important to state that Google doesn't recommend BoringSSL for general use, because there are no guarantees of API or ABI stability.

Having decided all the libraries we wanted to use, we only had to choose the algorithms that we wanted to test on the different devices.

3.3 Algorithms

The main objective of our app is to try as many algorithms as possible so we had to decide on symmetric, asymmetric and hashing algorithms.

3.3.1 Symmetric algorithms

As symmetric algorithms we have decided to use AES(Advanced Encryption Standard) with different block cipher modes of operation. All of these variations have been implemented using a 128-bit key.

AES-CBC: In CBC (Cipher block chaining), each block of plain text is XORed with the previous ciphertext block before being encrypted. This means that each ciphertext block depends on all plaintext block processed up to that point[Thi10].

AES-CTR: In CTR (counter mode), the encryptor generates a unique per-paquet value, and communicates this value to the decryptor[04].

AES-GCM: GCM (Galois/Counter Mode) is used to provide privacy and encryption. GCM maintains a counter; for each block of data, it sends the current value of the counter through the block cipher[autp].

AES-OFB: OFB (Output feedback) mode makes a block cipher into a synchronous stream cipher. It generates keystream blocks, which are then XORed with the plaintext[Bel].

It is also important to remark that not every library supports these algorithms. At the end of the chapter there is a table summarizing how is every library implemented.

3.3.2 Asymmetric algorithms

We also wanted to test asymmetric algorithms on our app. We have decided to implement RSA, Diffie-Hellman and Elliptic Curve Diffie Hellman.

RSA: RSA is a widely used algorithm used for security encryption. This algorithm is based on the difficulty of the factorization of two large prime. We found that it would be interesting to measure the performance of the CPU with such a demanding process. For our tests, we used OAEP padding with SHA-1 as hash function, as this is the most used variant and a 2048-bit key. Due to the own limitations of the algorithm, we will always encrypt a 128 byte string.

Diffie-Hellman: In the future we will also refer to this algorithm as "DH". Diffie-Hellman is a key-exchange method that allows two parties to generate a shared secret key over an insecure channel. This key can later be used to encrypt a symmetric key cipher[auta] In our tests we used the 2048-bit MODP group as established in RFC 5114[aut08].

ECDH: Elliptic curve Diffie Hellman is a variant of Diffie-Hellman where we generate the shared secret with the help of an elliptic curve public-private key pair. To generate our key we used the curve defined as *secp256r1*.

3.3.3 Hashing algorithms

Ultimately we have our hashing algorithms. For this type of functions we only implemented one type, MD-5, which we tested in different blocksizes.

MD5 is a hash function that produces a 128-bit value. We are aware of the numerous vulnerabilities that this algorithm presents. But we wanted to use it just to research the performance of the different devices while hashing; we do not recommend to implement it in any modern app.

3.3.4 Summary

Having explained all the different libraries and algorithms that our app implement, we consider important to clarify this. The next table summarizes this chapter:

RSA	Asymmetric cryptography	Encryption/Decryption
DH	Asymmetric cryptography	Key agreement
ECDH	Asymmetric cryptography	Key agreement
MD-5	Hash algorithm	Hash generation
AES-CBC	Symmetric cryptography	Encryption/Decryption
AES-OFB	Symmetric cryptography	Encryption/Decryption
AES-GCM	Symmetric cryptography	Encryption/Decryption
AES-CTR	Symmetric cryptography	Encryption/Decryption

Table 3.1: Classification of the different algorithms

	Bouncy Castle	mbedTLS	wolfCrypt	OpenSSL	BoringSSL
RSA	✓	✓	✓	✓	✓
MD-5	✓	✓	✓	✓	✓
DH	✓	✓	✓	✓	✓
ECDH	✓	✓	✓	✓	✓
AES-CBC(128)	✓	✓	✓	✓	✓
AES-CTR(128)	✓	✓	✓	✓	✓
AES-GCM(128)	✓	✓	✓	✓	✓
AES-OFB(128)	✓	Not provided	Not provided	✓	✓

Table 3.2: Algorithms used with the different libraries that will be later used to perform the benchmarks.

As we can see, AES-OFB is not supported by mbedTLS and wolfCrypt, so our tests will not include this AES variant.

Having explained how our app work, we would now like to explain how the tests were done.

4 Setting up the benchmarks

We already know how our app works and which libraries and algorithms we will use but now we have to see how we will measure the performance of the different devices.

4.1 Tests

The app supports right now 2 different tests, the so-called “Complete Test” and “Special Test”. In this section we will focus on explain how these tests work.

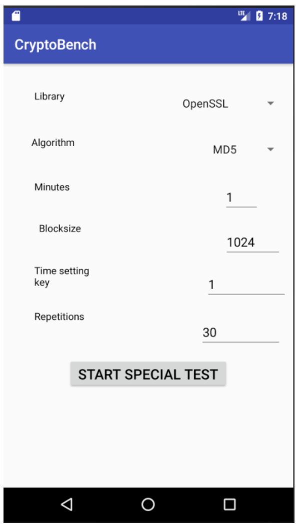
In the “Complete test”, the app will execute every algorithm from every library a finite number of times with all possible block sizes (128, 256, 512 and 1024 bytes). The user can establish the number of times they want to execute an AES, hash, key agreement and RSA operation. There is also a repetition value if it was necessary to repeat the whole test process. N is the total number of tests performed, AES, RSA, MD5 and KeyAgreement the user-given parameters and Repetitions the general value of repetitions. The total number of operations performed in this test is then defined by the next formula:

$$\begin{aligned} N &= ((\text{AES} \times 18 \times 4) + (\text{RSA} \times 5) + (\text{MD5} \times 5 \times 4) + (\text{KeyAgreement} \times 10)) \times \text{Repetitions} \\ &= ((\text{AES} \times 72) + (\text{RSA} \times 5) + (\text{MD5} \times 20) + (\text{KeyAgreement} \times 10)) \times \text{Repetitions} \end{aligned}$$

With this quick benchmark we can already test several aspects: first of all, we can see that every library/algorithm pair is working correctly. Although this may not seem important, having so many algorithms and libraries and having to test them manually will take at the end a huge amount of time. Also we can see how much time the mobile phone needs to perform the complete benchmark and later we can compare this data directly with our devices. Whereas in the “Special Test” the benchmarking will be performed in a specfied amount of time, with the “Complete Test” we can test directly how much time do the app needs. Although it may seems that the difference is small, it really makes a big difference. First of all, because we can test more algorithms directly and in less, secondly because having two difference ways of performing the benchmarks will result in a better way of comparing the different libraries and/or algorithms and allowing the users to have more ways to perform the different benchmarks. Obviously this can result in the problem that every user performs the benchmarks in a different way and later on we can not compare, in a fair way, the different results but this problem, we think, should not really be tackled on the developer side but on the user side.

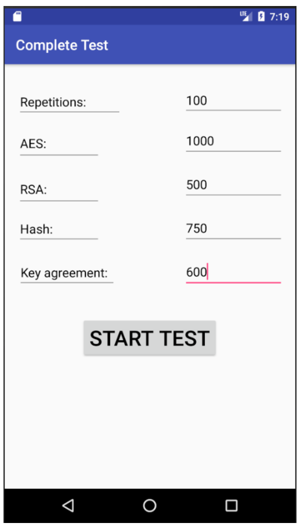
In the “Special Test” we will be able to perform a more specific test over the devices. Here we have to decide a library, an algorithm, a time of operation(in minutes), the block size, a time setting the key (in minutes) and the number of repetitions we want to do. We perform the benchmarks in that way so we can later calculate the different variations of results in a period of time. For example, we establish as “repetition value 1” and the “time of operation 30” and we will be encrypting for 30 minutes. The app will use to calculate the different results the value before the 30 minutes have started and right after these are finished. On the contrary, if we establish as “repetition value 30” and the “time of operation 1”, we will also be encrypting for 30 minutes, but every minute, the app will calculate the different results. We do it so we can not only calculate a byte/seconds ratio but we can also get the best, worst case, the mean, ... Apart from measuring the speed of the library/algorithm pair, this test measures the temperature of the CPU the same way it calculates the bytes/second ratio. Every time that the time of operation is finished, we will check the CPU temperature. As we explained before, the problems of several cryptographic benchmarking suites is that in the own process of the benchmarks, two things were being measuring at the same time, so for really fast algorithms, we would, at the end, not know what we were really measuring, it can be the own encryption function or the function that calls the clock to check the time. In our approach, we tackle this problem by generating two different threads. In the first one, the app will be executing the encryption process and in the second thread, we have a variable that will change after every minute of the time of operation has passed. In that way, the function will always be measuring only the encryption process. The disadvantage of this process is that generating two different threads has a consequence on the performance of the device, as it will not be using the 100% power of the CPU. Although we have already seen in 2.3 that the CPU will never use the full power in the process, we are adding more counterparts to the operations. If we were only performing only one operation (i.e. one encryption), the difference will be very small and we could ignore it, but as we are performing several thousands of them, at the end the results would be very affected. In that way, we can calculate how the temperature of the CPU varies during a cryptographic operation. The “time setting the key” will allow us later to establish the average time that a library needs to initialize the classes, setting the parameters,... if we needed.

It is important to remark that some algorithms will always use the same parameter. For example, with the purpose of avoid possible problems, RSA will always use a 128 bytes string.



Library	OpenSSL
Algorithm	MD5
Minutes	1
Blocksize	1024
Time setting key	1
Repetitions	30

Figure 4.1: UI of the "Special Test"



Repetitions:	100
AES:	1000
RSA:	500
Hash:	750
Key agreement:	600

Figure 4.2: UI of the "Complete Test"

To start a test, the user can do it through the integrated UI, but also he can do it through ADB if he wants to have more information, as the app is prepared for it.
To start a complete test, it would be necessary to use this bash line:

```
$ am start -n com.example.CryptoBench.sac/.CompleteTestActivity
-e test 1 -e aes 20 -e hash 7 -e dh 5 -e rsa 15
```

And for a special test the terminal command is:

```
$ am start -n com.example.CryptoBench.sac/.ConcreteTest -e lib
BoringSSL -e algo DH -e min 1 -e  blocksize 128 -e key 1 -e rep 2
```

To analyze the benchmarks, we stored every result after every operation time has passed in a text file that the user can access in the device. At the same time, this text file will be sent via e-mail to our own account. Although we didn't use this option, as we always had the devices, this option can be used to analyze different results from different parts of the world. We find this interesting as we could see how the users use the app and which are the results on different devices. Also, having access to a text file with the different results makes easier the process of analyzing the data. We used for example a Python script that will calculate the different statistics for the data that we found more interesting but different users can decide what they want to do with the data.

If more information of the app is needed, the source code and the apk[1] is available at `https://github.com/juanpas97/CryptoBench`.

4.2 Devices

Also, it is important to explain on which devices will the test be done, because with this information we can estimate how good the performance of the different CPUs is.

Our first researched mobile phone is a **Samsung Galaxy Note 4**. It was released on 2014 with Android KitKat 4.4.4. It uses a Qualcomm Snapdragon 805 and 3 GB of RAM. At the time of release, this was a high-performance mobile phone, so it is interesting to see how the general quality of this device has changed after all these years[autm].

The next one is a **Samsung Galaxy S4**. This is an older phone, as it was released on 2013. The device uses a Qualcomm Snapdragon 600 Quad-Core, an older version of the processor that the Samsung Galaxy Note 4 uses and 2 GB of RAM. It also uses a more aged version of Android, as it has Android 4.2.2 Jelly Bean. In general, this device should present worse results than the previous one[autn].

We have also used a **Motorola Moto X Style**. This phone should get the better results, as it is the newest one and it uses high-performance components. It was released on July 2015, it uses a Snapdragon Qualcomm SnapDragon 808 and 3 GB of RAM. This is also the mobile

[1]Android PacKage (APK) is the package file format used by the Android operating system for distribution and installation of mobile apps and middleware.

phone that, in comparison with the other devices, uses the most recent version of Android, 7.0 Nougat[auti].

Our last device is a **HTC One X**. This is a low-performance device released on 2012 and it is the only one of our devices that doesn't use a Qualcomm CPU, instead it uses a NVIDIA Tegra 3, 1 GB of RAM and Android 5.1.1.

The next table summarizes the characteristics devices to have a better lookup at the differences:

Device	CPU	RAM	Android Version
Samsung Galaxy Note 4	Qualcomm Snapdragon 805	3 GB	4.4.4
Samsung Galaxy S4	Qualcomm Snapdragon 600	2 GB	4.2.2
Motorola Moto X Style	Qualcomm Snapdragon 808	3 GB	7.0
HTC One X	NVIDIA Tegra 3	1 GB	5.1.1

Knowing how the test was done and in which devices the tests were performed, now we can present our results.

4.2.1 General performance of the devices

As we have explained in the section before how the devices should generally behave, we will nevertheless take a closer look to them by using already done benchmarks. For these benchmarks, we will use data from an Android App called "Antutu" [unk17]. This app performs several benchmarks to the device and measure the general performance of the CPU. We can use these results to have a quick idea about the power of the different mobile phones.

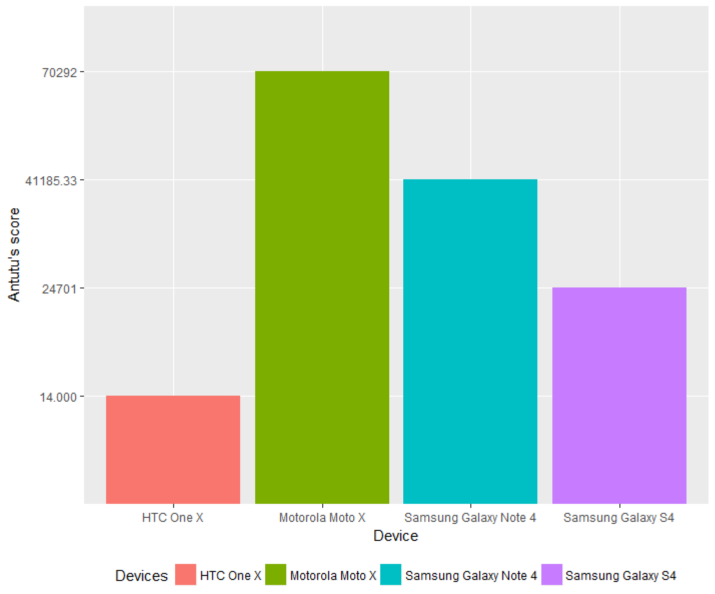


Figure 4.3: Antutu's score of the different devices

4.3 Assumptions over the algorithms

Before explaining the different results that we encountered, we would like to state what we expect from the results due to the own characteristics of the algorithms and the tests. First of all, as we have explained, we will perform the different tests with different block sizes for each algorithm. We will start with 128 bytes string and finish with 1024 bytes. With this big difference between the strings, we expect differences in the results that will be correlated with the different block sizes, so the results will be better for a 128 bytes and worse for a 1024 bytes string. Also, for the different AES modes we should not expect a big difference between the encryption and decryption process, as using the same key for the process had to diverge in similar results. Also, AES-CTR and AES-GCM should have worse results than AES-CBC and AES-OFB, as with these modes, more data is encrypted due to what it is explained in 3.3.1. Secondly there shouldn't be any big difference in the results of MD-5, as the algorithm will divide every string in 16 bytes blocks, so at the end, will always be the same. Where we really expect big differences in the results is in the RSA algorithm. The encryption process should present better results than decrypting, as the mathematical processes of the decryption require more computational power. We expect the same as with Diffie-Hellman and Elliptic Curve Diffie-Hellman. ECDH makes use of algebraic curves and it needs more power to the calculate the shared key. It is important to remark that although this makes the processes slower, it really has an effect of the security of the algorithm, making them more secure to brute-force attacks.

5 Results

It is important to explain some concepts before we go straight into the results and in which conditions were the tests performed.

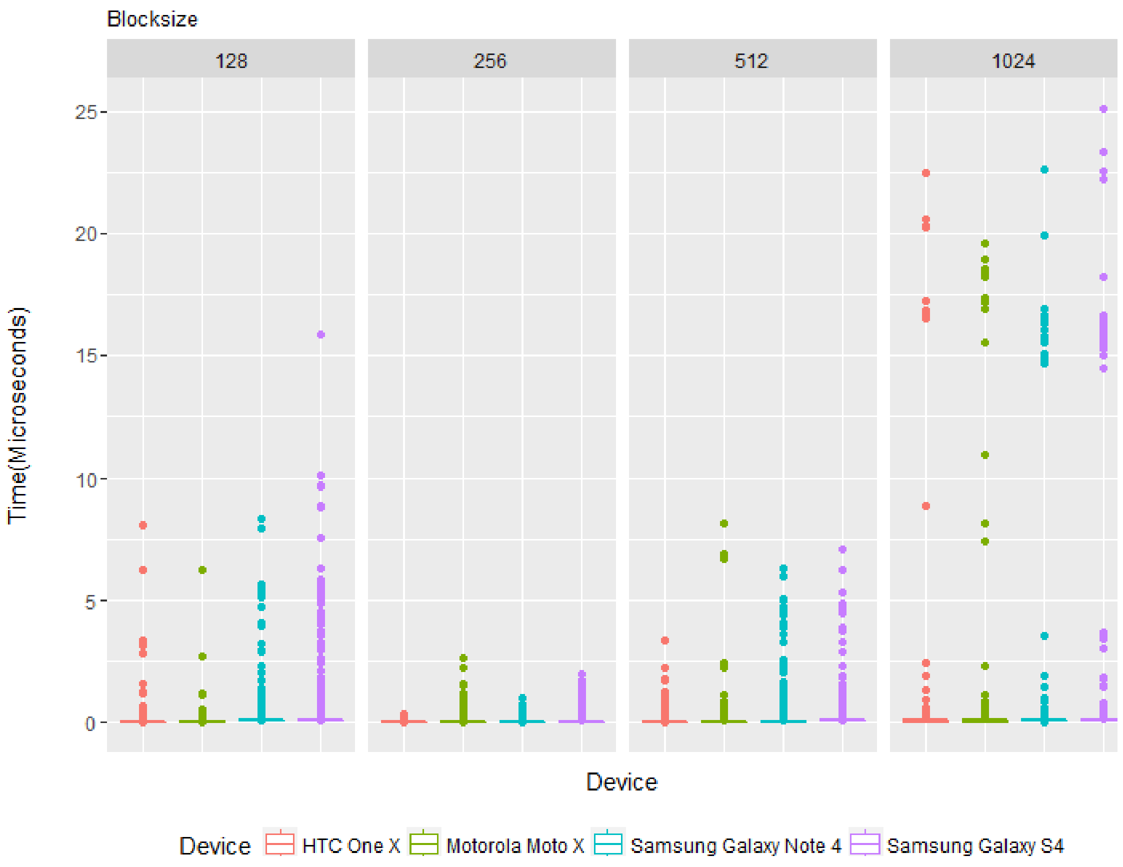
All the tests were performed under the same conditions to ensure the highest fidelity and consistency.

Every algorithm was tested a total time of 30 minutes, with a 1 minute operation time[1] and between the tests we waited always minimum 15 minutes so the devices could recover from the previous test and could get back to its original temperature, so we could also see if there is a "cold start" in the libraries. In this way, we will get significant data to see the performance of every library/algorithm pair. In the different sections we will be able to analyze the results with the help of graphs the most significant differences. Every corespondent result is in the appendix if more specific data are needed.

5.1 JNI call overhead

In every library used with the exception of Bouncy Castle we are using C libraries. This means that the app has to handle the native code. For this task, the variables have to be sent through the JNI to the C code and then we have to recover the results from them. Although this time is minimal, it also has to be taken into account as the results will always be slightly slower because of this aspect.
To test how much time does Java need to handle this "data transfer" we have developed a little app called C-Transfer where a random string (we used the Java primitive type *String*) of different block sizes (128, 256, 512 and 1024) will be sent to a C function. In this function, we will copy the content of the string to a new one, then this new string will be sent back again to Java. To perform the tests and get a considerable amount of samples we performed this action 100.000 times in every device. We did not consider that this part should be integrated into the "CryptoBench" app because this is something inherent to every C library. If you want to use a C library in an Android app, this time is always present and cannot be reduced. Also we have to consider that the results are really small and unless we want to be constantly performing a cryptographic operation (and sending and receiving data from the native code), the results of the benchmarks will not be influenced. The results are presented in the graph below:

[1]30 minutes encrypting and 30 minutes decrypting in the case the algorithm needs it



As we can see, the Motorola Moto X and the HTC One X present the best performance with this type of test. It is surprising that the HTC One X gets better results than the Samsung Galaxy S4, because the HTC device has lower specifications. The time is presented in microseconds so this time (as we will see in the following sections) does not mean a significant difference if a developer just will perform one operation, but for long-time cryptographic processes, this can results in losing several seconds. One of the most interesting aspects is the variance that this operation produce in the different devices. Because of this, we can not give a specific amount of time but we only can see a range of the time this operation needs. To demonstrate that this time is insignificant to the final results of our algorithms, we take one device, a library, an algorithm and a string(in this case the Motorola Moto X, OpenSSL and a 1025 bytes string) and we check it with the worst and the best time of our C-Transfer app. To encrypt just one string with the best result, we will need around 16300,0141 milliseconds and with the worst result will be 16300,125 milliseconds. The overhead of this data transfer will only mean (max.) in a 0.01% of the total time elapsed.

Knowing all of this we can now go deep into the results we achieved from the different tests.

5.2 Performance Test

5.2.1 AES-CBC

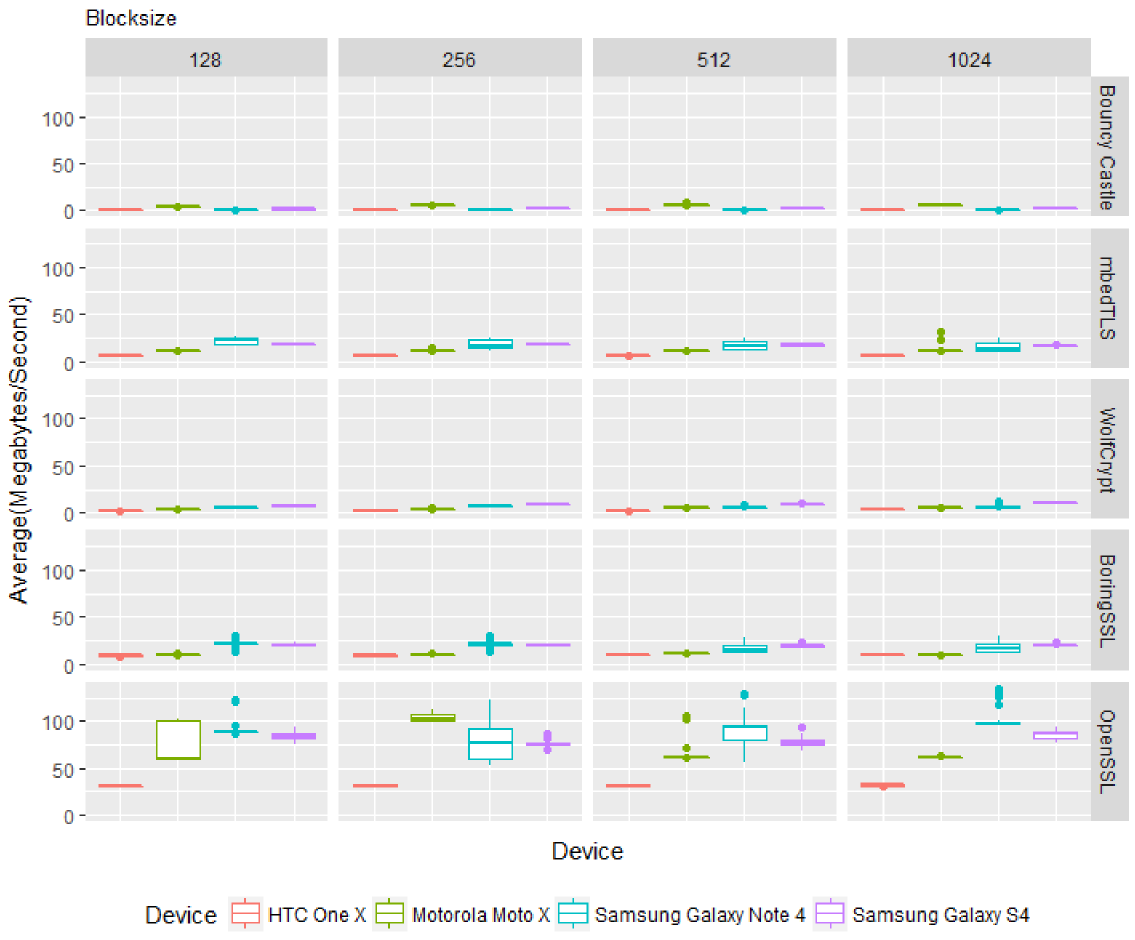


Figure 5.1: Results of AES-CBC Encryption

We will start with the results of the AES-CBC algorithm. First of all, it may be surprising that there is no major difference between the performance of the different libraries with every block size. The difference between encrypting a 128 bytes or a 1024 bytes is minimal, although with longer block size all the libraries have a little better performance. It is also astonishing to see (and in the next results this will be confirmed) that Bouncy-Castle will always get one of the lowest results while performing AES operations, meaning that executing the tests with the Java security provider will be worse than using a C library. For this algorithm, Bouncy Castle, mbedTLS and wolfCrypt get the worst results. They will always encrypt less than 25 Megabytes pro second. BoringSSL will sometimes break this barrier but without any doubt, the best library to use AES-CBC is OpenSSL, although the results usually have more variance, the bytes/second rate will always be higher. It is also important to remark how the devices have behaved during the test. As we predicted, the HTC One X will always get lower numbers than the other devices, this is normal due to the low-performance nature of the device. What it is more surprising to see is that the Samsung Galaxy S4 sometimes gets better results than the Motorola Moto X, although this a better device. Even in OpenSSL, where the devices get more dispersion of the results, the Samsung Galaxy S4 gets more consistent results than the Motorola Moto X and the Samsung Galaxy Note 4. In general, the Samsung Galaxy Note 4 will get better results than the Motorola Moto X. This is also a bit strange as the CPU and the Android version of the Samsung device are a bit outdated. Although using OpenSSL, the Samsung Galaxy Note will have less consistency with the results, as they have a bigger variance.

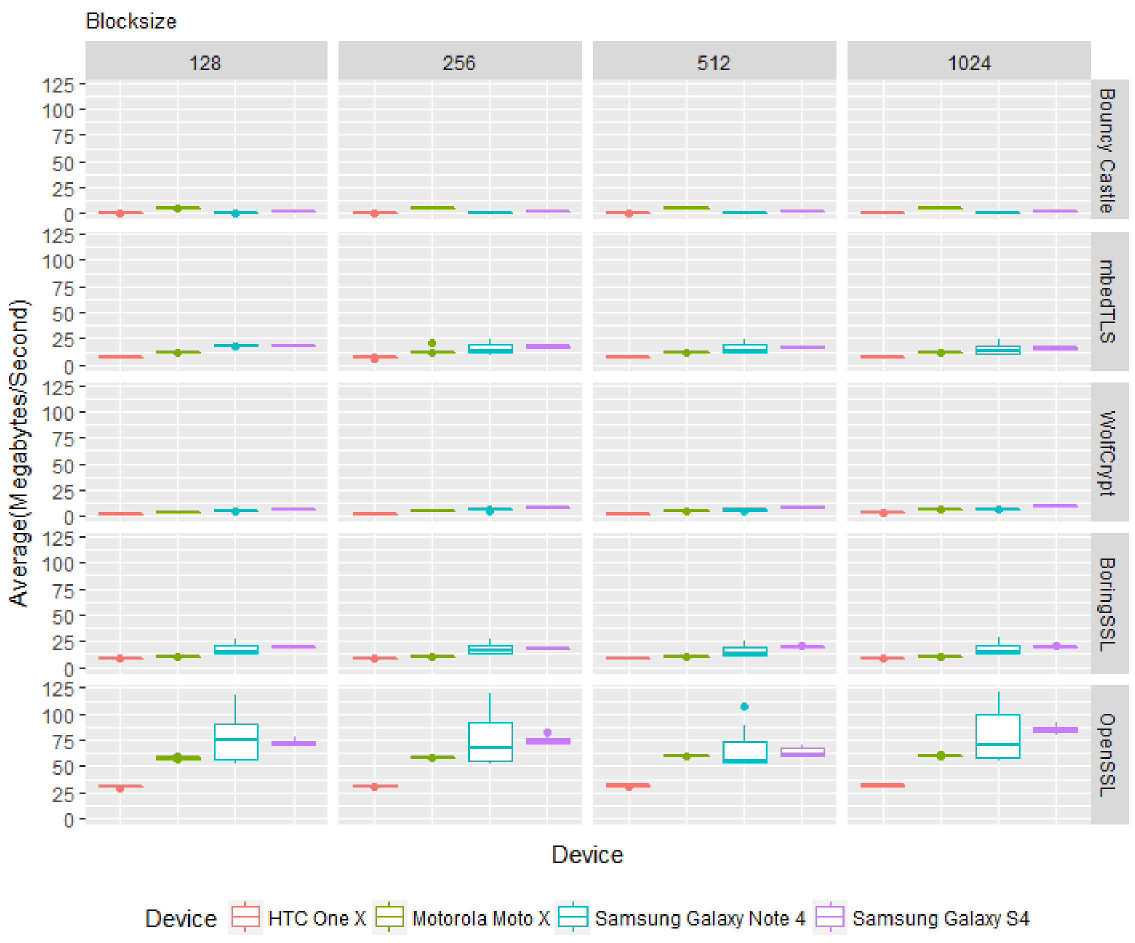


Figure 5.2: Results of AES-CBC Decryption

The first aspect to comment about this operation is that the decryption of every algorithm will always be highly correlated to the encryption operation of the same library. This means that we will not see very big differences between the bytes encrypted and decrypted of the different library/algorithm pairs. For AES-CBC we have almost the same results that in the decryption, this means, being OpenSSL the best library to perform this operation and the HTC the worst device. The biggest differences here is that the Motorola device has less variance while decrypting and on the contrary, the Note's variance will grow. Due to the growing variance of the device, it can get to decrypt more bytes and get peaks of almost 125.000.000 million of bytes. Also with the decryption operation the Samsung Galaxy S4 performs better than the Motorola Moto X, and although its results have more dispersion, they will always be higher than the ones from the Moto X.

5.2.2 AES-CTR

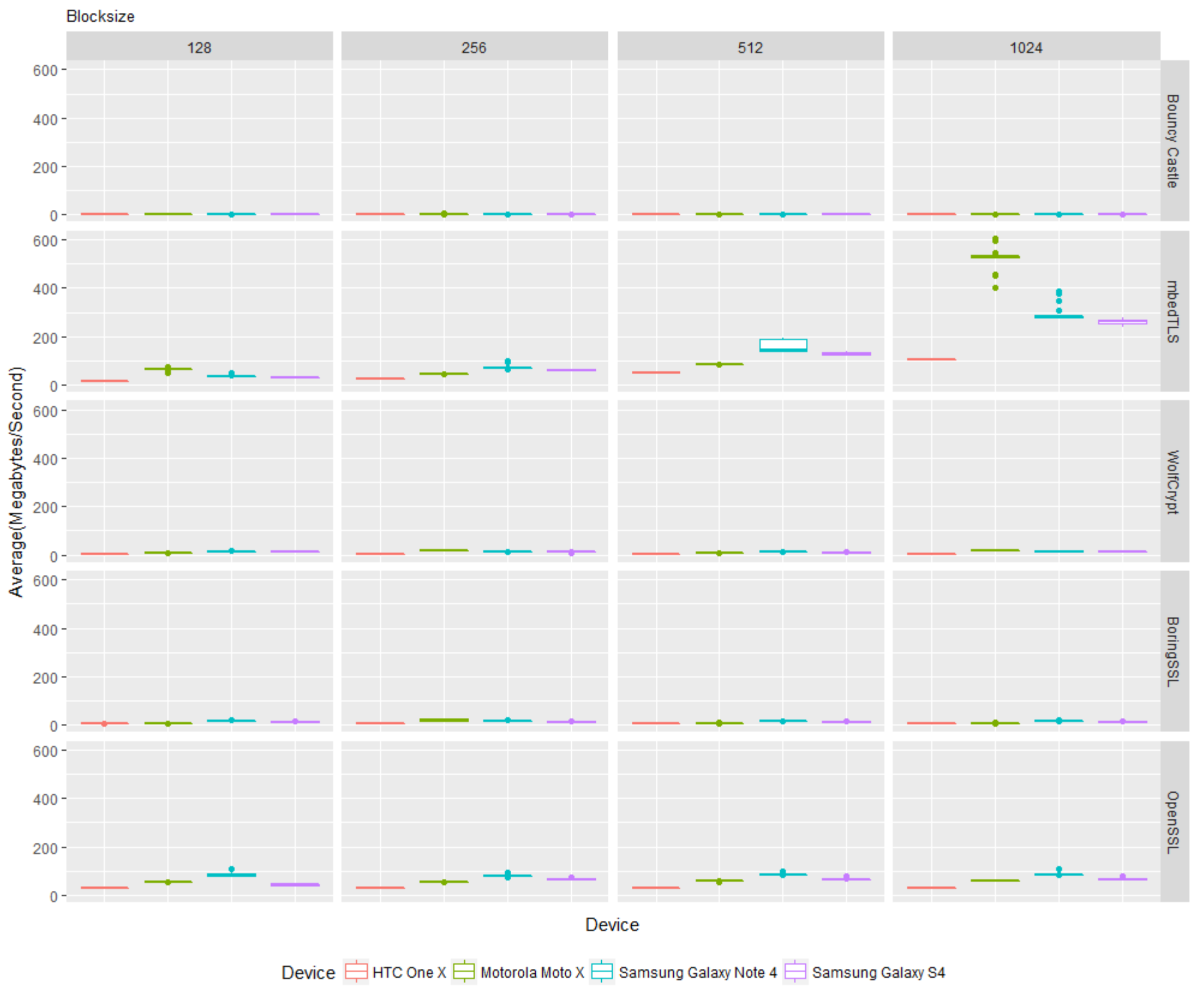


Figure 5.3: Results of AES-CTR Encryption

What first comes to mind when we see the results of AES-CTR is that they are more accurate than for the AES-CBC, as we can observe there is not so much variance as in the previous algorithm. The results here are very similar to the ones observed before: Bouncy Castle keeps being the worst library in terms of bytes/second but the results of wolfCrypt and BoringSSL are pretty close to it. In this library there is not much difference between the different devices, but HTC One keeps showing a lower performance than the other devices. In spite of these similarities, there are important differences that we need to point out. First of all, we can see that OpenSSL is not the best library for AES-CTR. Its results keep being better than the libraries previously mentioned but mbedTLS shows a better performance. The second main difference is the big difference between the Motorola device and the others in mbedTLS. Apart from the 1024-bytes block size, the performance of the Motorola and the Samsung Galaxy Note 4 are very similar, but in the 1024-bytes block size, the Motorola Moto X performance grows until reaching over 500 Megabytes/second,one of the biggest encrypting rates.

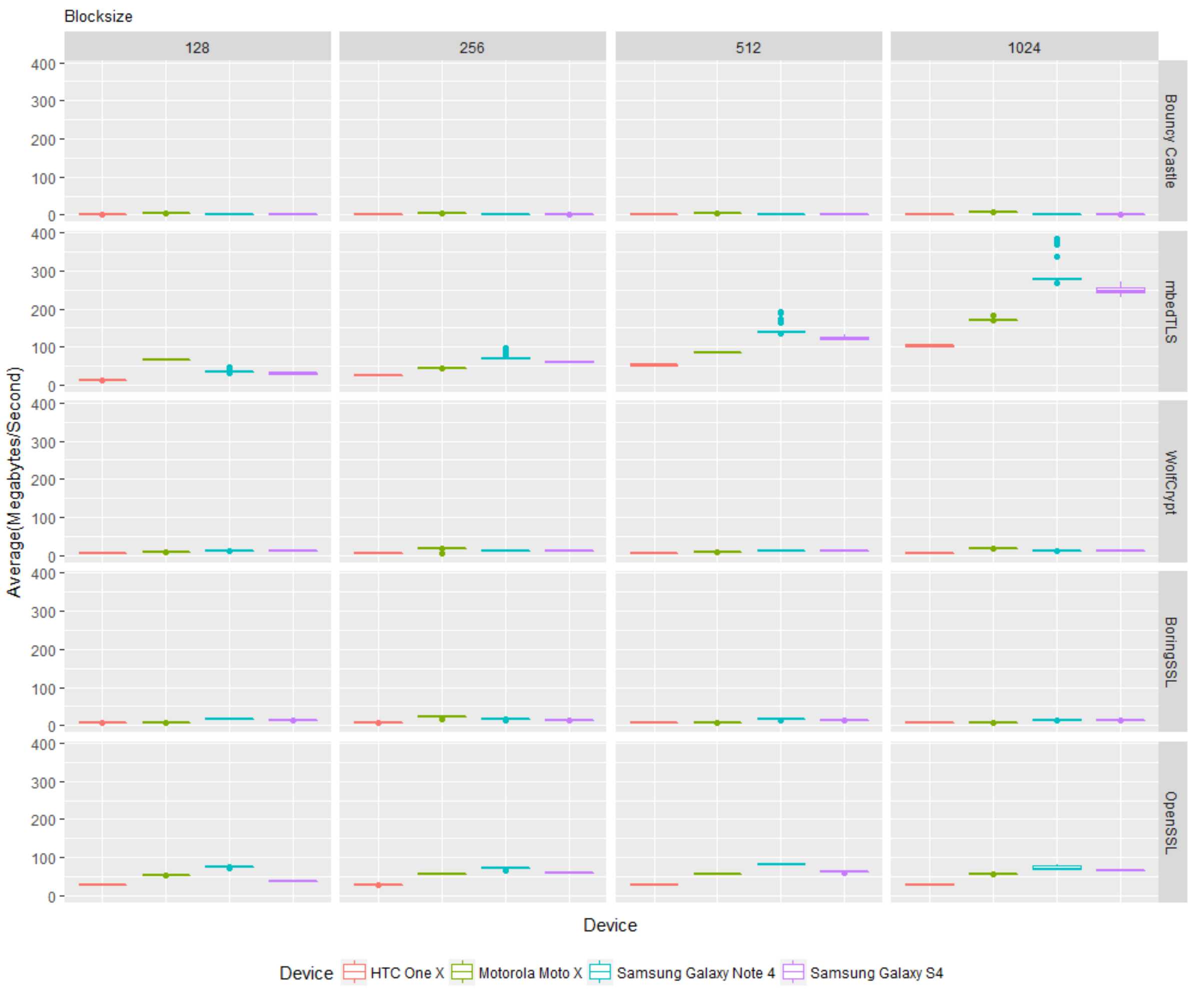


Figure 5.4: Results of AES-CTR Decryption

As in the other algorithms, we can see that there are not big differences between the encryption rate and the decryption rate. This operation is even more accurate than the encryption, as the results present less variance between them. The most surprising aspect of this algorithm, is that the big encryption rate of the Motorola Moto X in mbedTLS is not correlated to the decryption rate. In fact, the Samsung Galaxy Note 4 achieves better results in decryption operations. For the developers, it should be important to know this little difference because this could mean an improvement boost if they only pretend to decrypt files on a device.

5.2.3 AES-GCM

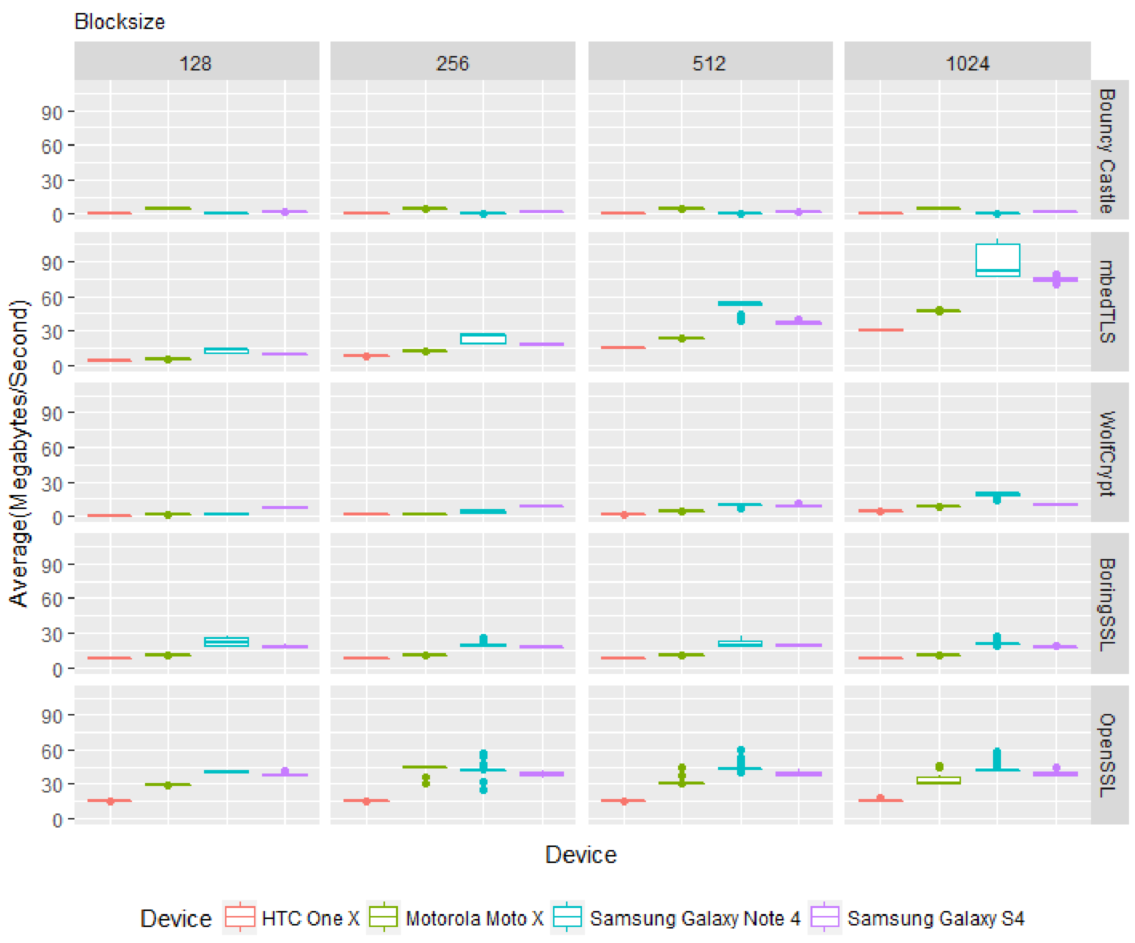


Figure 5.5: Results of AES-GCM Encryption

In this algorithm we can see the low general performance of the algorithm in every device and library. In this case, the bytes/second rate is generally smaller than in the other researched operations. As we could see in the other two graphs, OpenSSL would generally be the most powerful library and its numbers usually surpass the other libraries. But in this case, we can clearly see that mbedTLS is the best library to perform AES-GCM with 512 or 1024 bytes block size strings, as its numbers in some cases double the results of the other libraries. It is also remarkable that for almost every device, the variance is minimal in every library with the exception of the Samsung Galaxy Note. Although it has a lot of atypical values (which show an irregular performance), it is clearer in this algorithm that the Samsung high-performance device keeps being the best mobile phone to perform AES decryption.

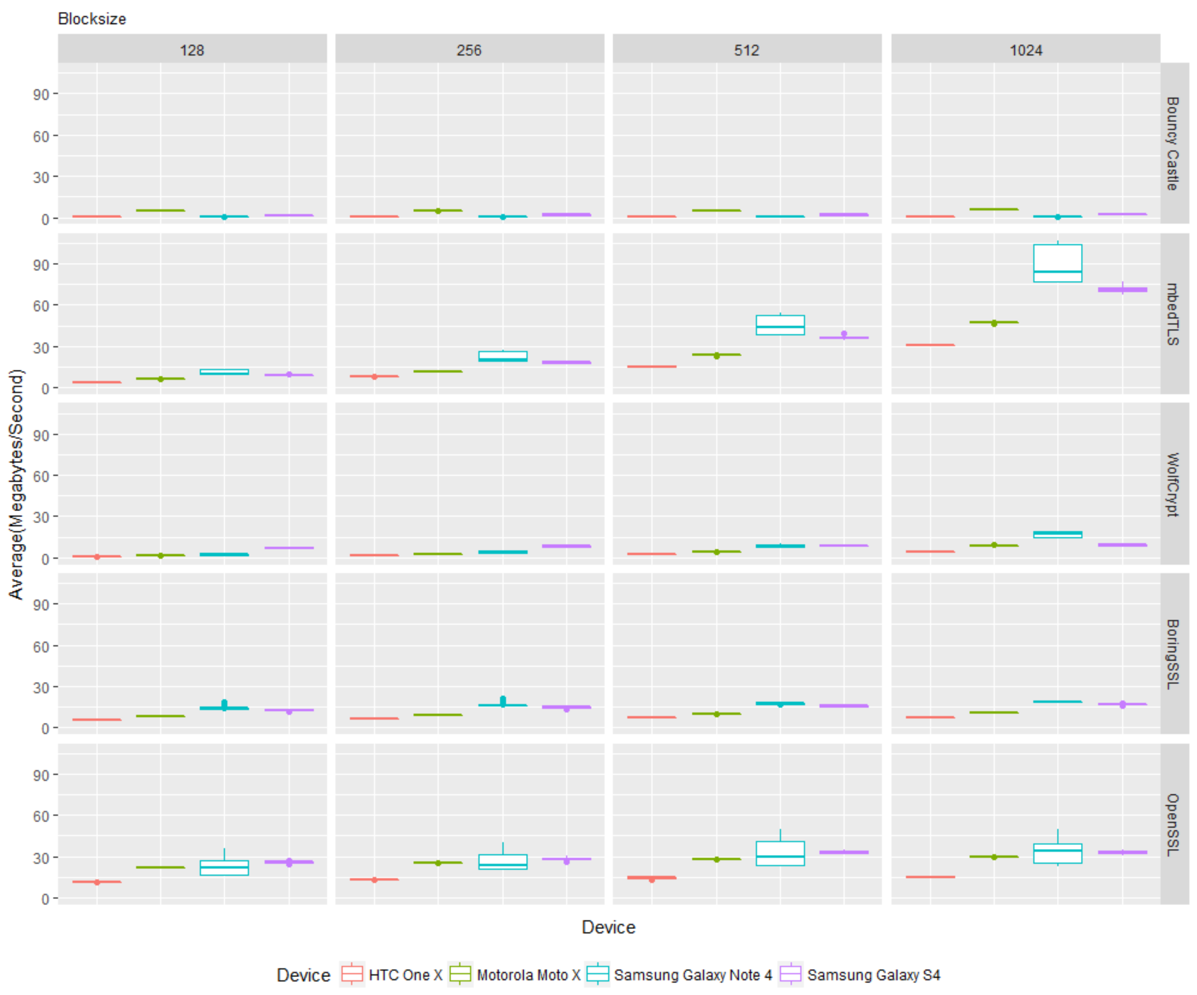


Figure 5.6: Results of AES-GCM Decryption

As we have previously observed, we can still see how strong the encryption and decryption operations are correlated. In this case, the differences between both operations are almost inexistent. The devices still show a very small variance with their results but in this case, the Samsung Galaxy Note 4/OpenSSL pair results are more diverse. In spite of this observation, it is still the best device to perform this operation. It is also remarkable to say that for wolfCrypt and Bouncy Castle, the results of the different devices for the different block sizes are very similar. Here we don't have a big difference so developers who want to have a steady rate for every device could use both of these libraries, although this would mean a lower performance than the other alternatives.

5.2.4 AES-OFB

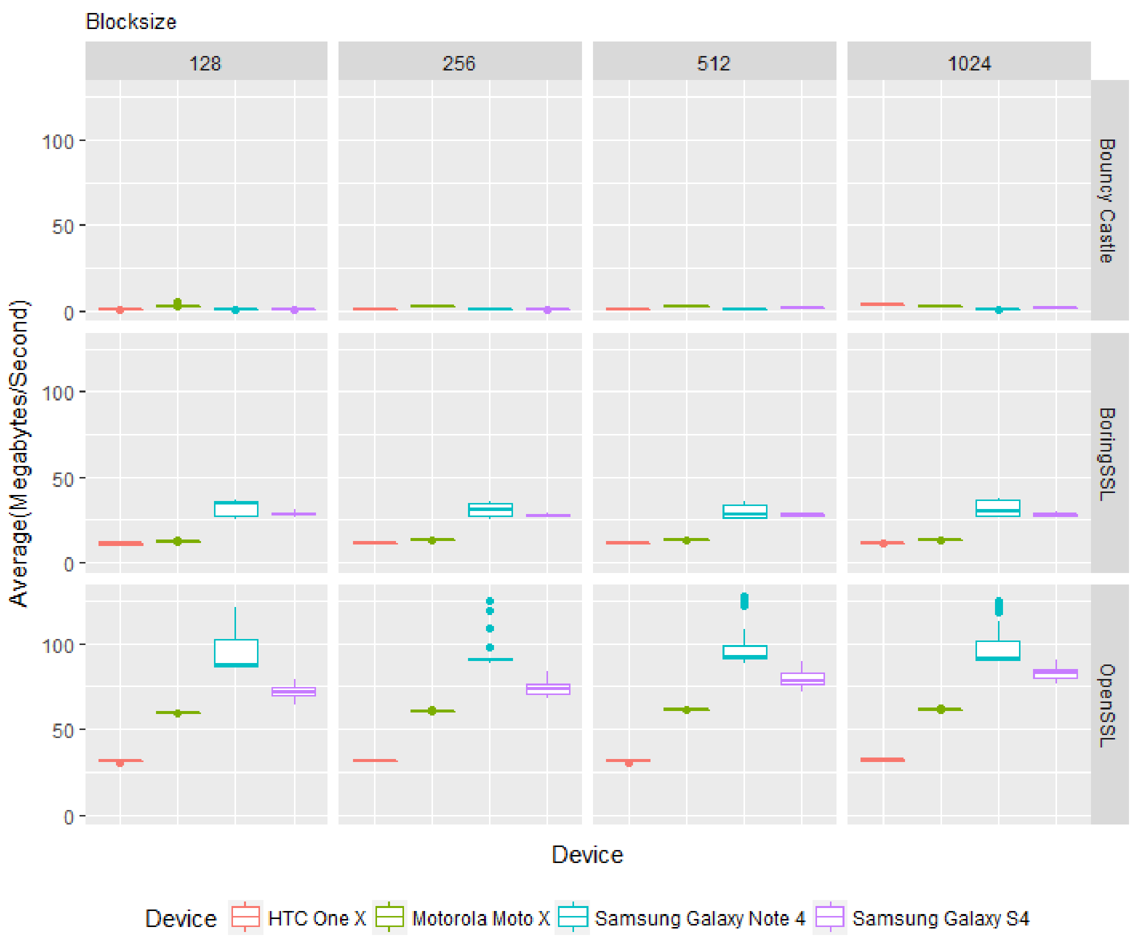


Figure 5.7: Results of AES-OFB Encryption

The first aspect to comment of this graph is, as said before, the absence of mbedTLS and wolfCrypt. As these libraries do not support this algorithm we are not able to investigate how they would perform on the different devices. This is important for developers who want to use different AES methods on their apps, as the size of the final APK will increase as they have to include more libraries if they originally wanted to only use WolfCrypt or mbedTLS. Coming back to the results, we can here clearly see how bad the performance of Bouncy Castle is. While in the other libraries the results of the encrypting operation is (minimum) over 10.000.000 bytes/second, only in the best cases the Java library will reach 3.000.000 bytes/second. The only good aspect of this is, as we previously said with AES-GCM decryption is that every device perform almost the same, with only small differences between the devices. It is also surprising to see that there is no big difference between the different block sizes in BoringSSL and OpenSSL. The results of the different test show a constant byte/second rate. The Samsung Galaxy Note 4 is again the device that performs better the encryption operation, but it still shows this inconsistency in the results.

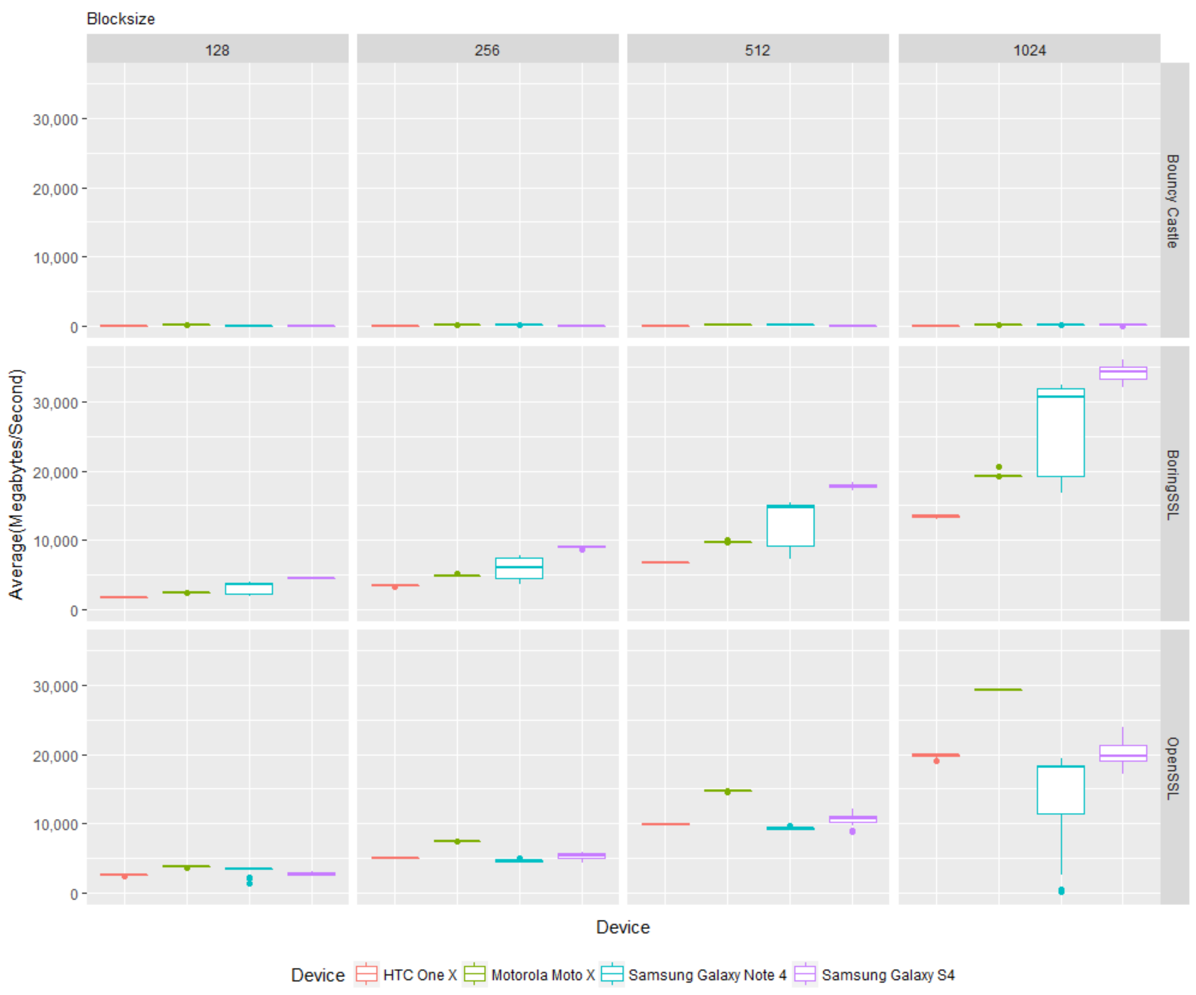


Figure 5.8: Results of AES-OFB Decryption

Without mentioning the typical aspects that we have already commented in the other decryption operations, what is more surprising to see here is how the Samsung Galaxy Note 4 is not the best device to perform this operation. In fact, it is not even the second best option to do it. Taking as reference the library that presents the best results (OpenSSL), we can see that the Motorola Moto X will not only get better results, but also it will have less variance than the Samsung Galaxy Note 4. The results of the Motorola device are clearly superior than the other devices and usually presents a 150% better rate. As second best device we can see that the Samsung Galaxy S4 performs a little better in OpenSSL. This device also performs better on BoringSSL, being this a big surprise as this device have clearly worse components than the other two. Last, what is most surprising about this algorithm is how the HTC One X performs. We have in the previous operations seen that the HTC device is clearly the one that presents the worst performance. Normally there is a big difference between this device and the others but here, we can see that it does not only get better results with respect to the previous decryption operations, but also it presents better results than high-performance devices as the Samsung Galaxy Note 4. The low-spec device will reach around 20.000.000 decrypted bytes/second in the best case, which is a very good result.

5.2.5 RSA Results

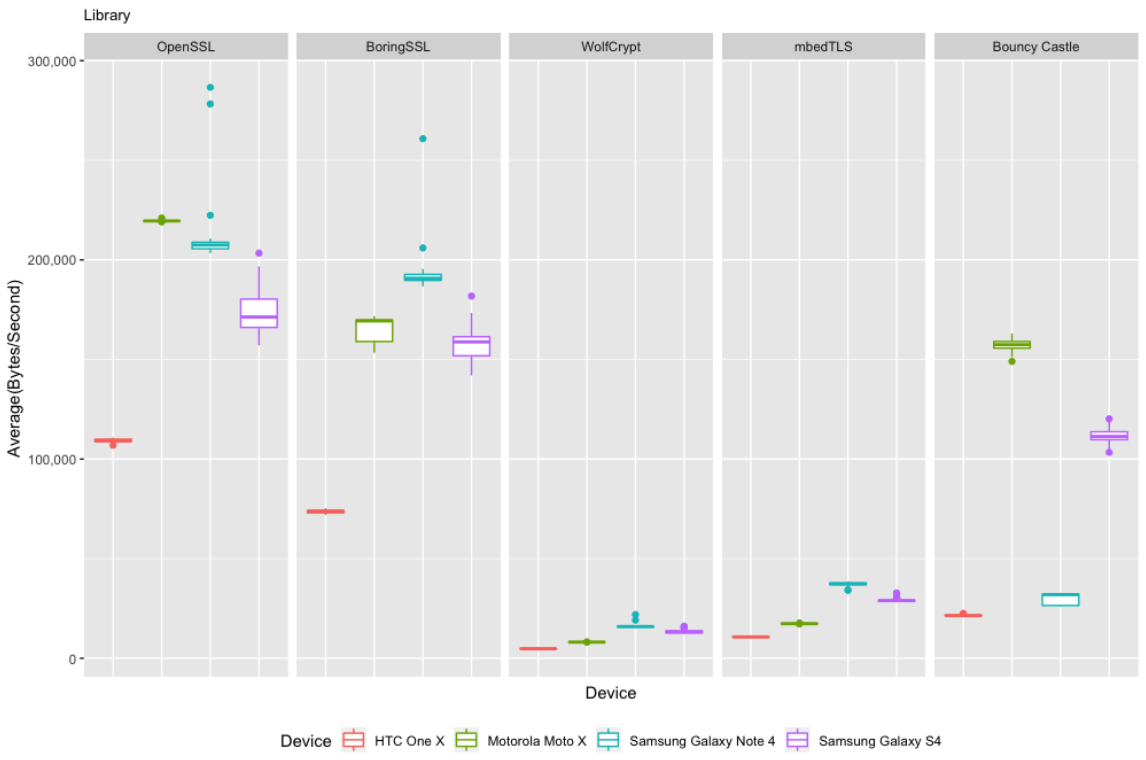


Figure 5.9: Results of RSA Encryption

As we have seen in the different AES operations there is significant differences between the libraries. In RSA, this trend is not interrupted. As it is being usual, OpenSSL is the best library to perform RSA encryption because its numbers are always over 100.000 bytes per second (inclusive in the worst device). The next best library is Boring SSL, which should not sounds strange as it shares the majority of the code with OpenSSL. What it is more surprising is the good performance of Bouncy Castle over several devices. In this case, the Java security provider performs better than mbedTLS and wolfCrypt, which both of them have similar results. Another aspect to have in mind is the variance in the Samsung Galaxy S4, whereas in the other devices the results do not present a significant variance on the results. The Samsung device does, specially when using OpenSSL. It is also important to point out the small differences that wolfCrypt and mbedTLS present between devices, as there is not a big difference between the different rates. Last, it is interesting to mention that the best device to perform RSA depends on the library that we are using at the moment. If we use OpenSSL (which would be the ideal option as presents the better results) or Bouncy Castle, we would use the Motorola Moto X. On the contrary, on BoringSSL, wolfCrypt or mbedTLS the best device to use is the Samsung Galaxy Note 4, again.

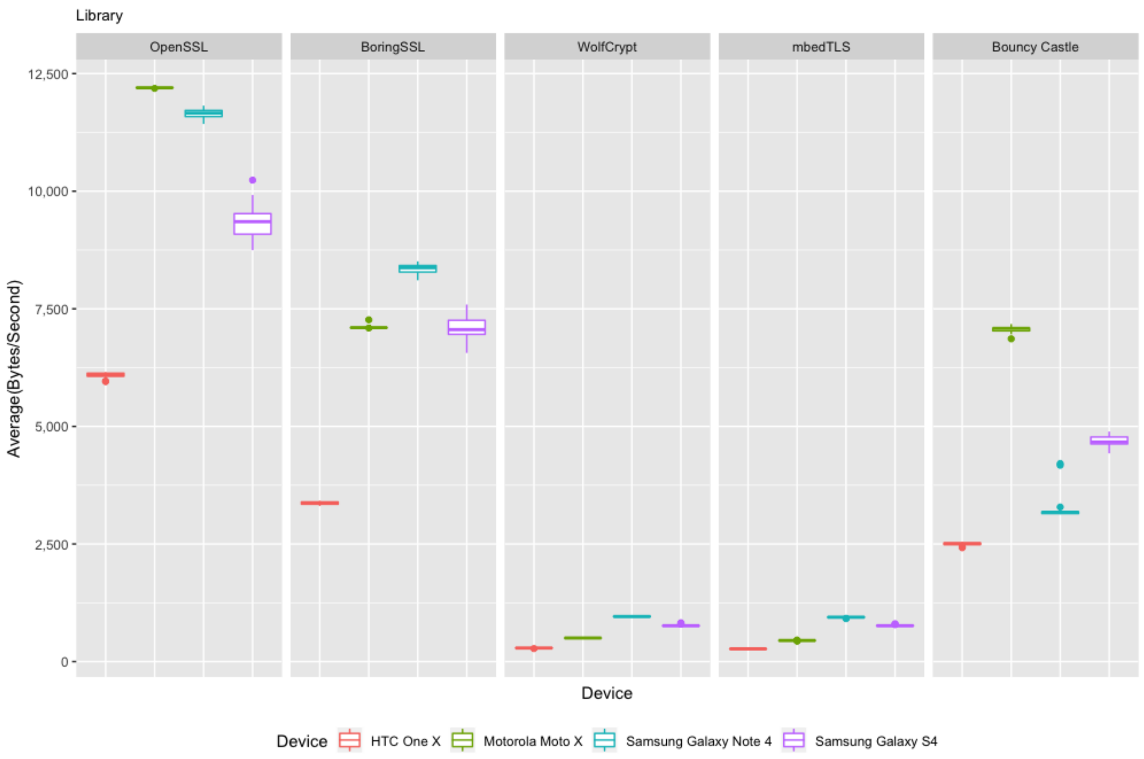


Figure 5.10: Results of RSA Decryption

For the first time in all the algorithms researched, we can clearly see that the encryption results do not correspond with the decryption results, as they are 94% slower. This is normal due to own nature of the RSA decryption operation. Apart from this, the results are highly correlated as the best libraries encrypting will also be the best libraries decrypting. We will have the same trend with the devices, the best device encrypting will be the best device decrypting.

5.2.6 MD-5 Results

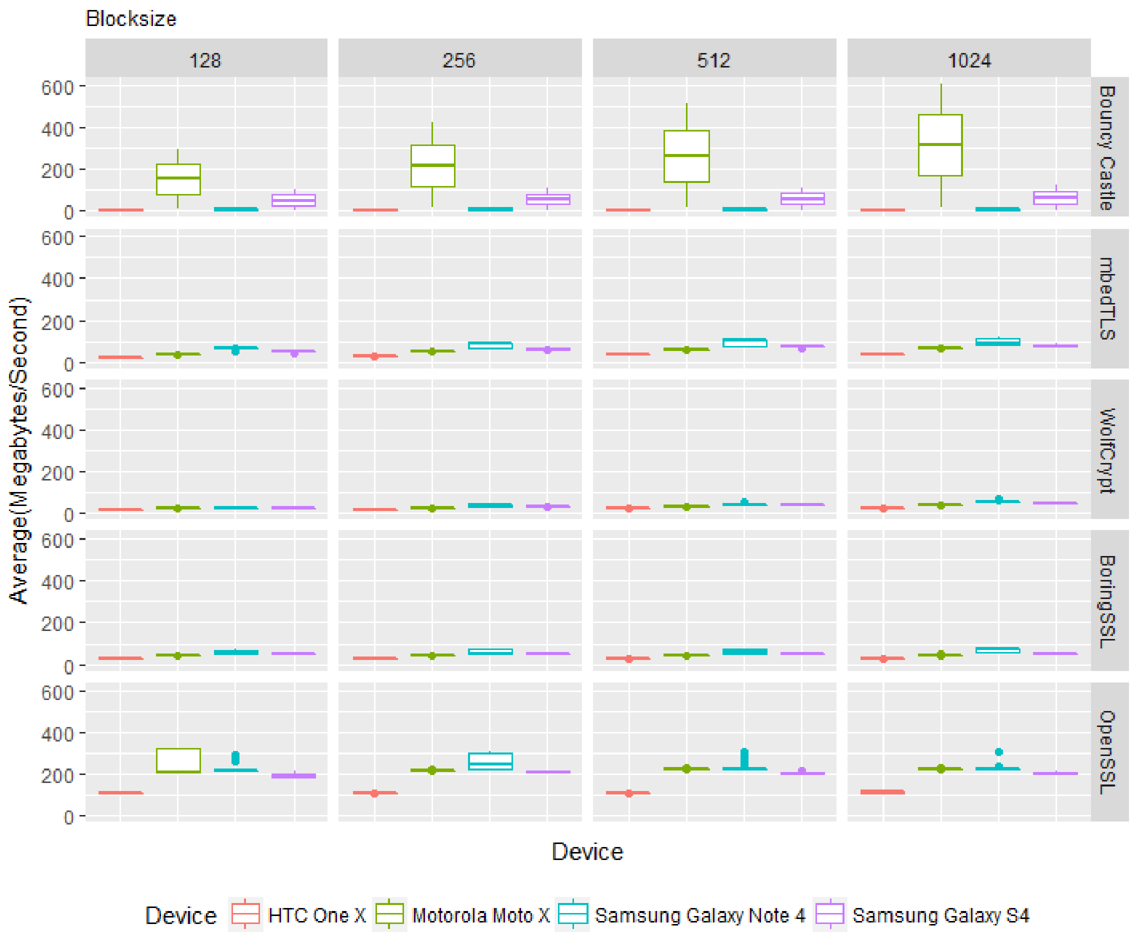


Figure 5.11: Results of MD-5 hash generation

Clearly we can here observe how for this algorithm every block size has the same performance in all the libraries. As we can see there is not big differences between hashing a 128 or a 1024 bytes string. This is normal because the algorithm will split every block size into 16-bytes block and then will perform the hash. Like in the previous tests, OpenSSL will perform the hash in a 200 megabytes/second rate. One of the most surprising result of this test is how the Motorola Moto X behaves on Bouncy Castle. It will get very good results (around 200 megabytes/second) but it presents a very big dispersion on its results, making it a very inconsistent device to perform MD-5. This variance is also presented in the Samsung Galaxy S4 but in smaller range. After pointing out the most surprising results we can see that no much has changed in relation with the previous tests. OpenSSL is the best library followed by BoringSSL and mbedTLS.

5.2.7 Key Agreement Results

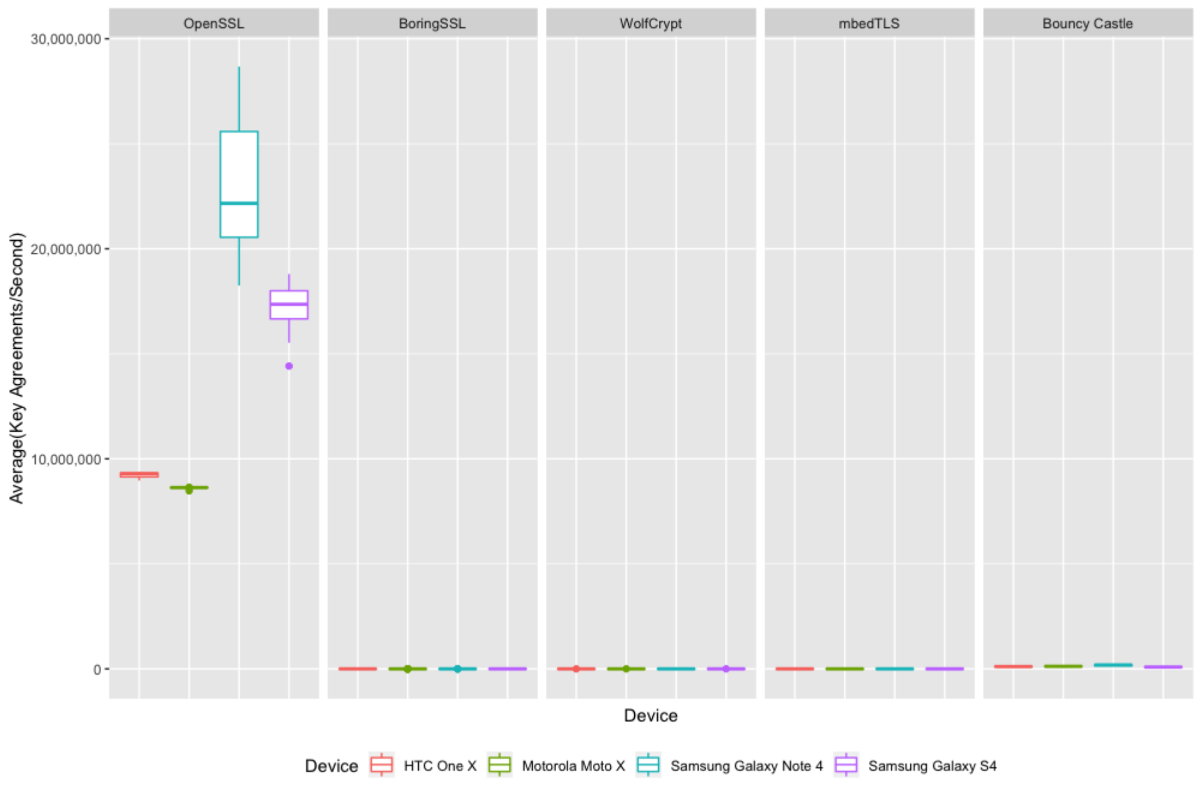


Figure 5.12: Results of DH Key agreement

For these algorithms we get very surprising results. As we can see, the difference between OpenSSL and the other libraries is so big that it makes the other results look insignificant. Although for this algorithm Bouncy Castle performs really good, as it can generate over 10.000 key agreements per second, the results for OpenSSL are so good on the different devices that if a developer would want to just implement this algorithm, it would be meaningless to use another library. Before explaining the other results, one of the possibilities that the other libraries get so low results may be because normally, the "generate key agreement function" of the libraries (apart from OpenSSL) are bounded to a time constant to avoid an overload of the CPU. In spite of this, it can be a good idea if we do not want to perform a lot of key agreements per minutes, but this will lead to bad results if we want so many key agreements/second as possible. In the case of mbedTLS, it is a really bad sign that the library is not able to just generate one single key agreement per second. This special case will later be researched in the "Temperature test"(5.4) to see the performance of the CPU using this library. wolfCrypt and BoringSSL also have lower results than OpenSSL. It may be surprising such a big difference between OpenSSL and BoringSSL (BoringSSL is the Google's fork of BoringSSL) but this can be produced due to the fact that OpenSSL allows two different ways of implement the Diffie-Helmman library (one with time constant and another without it) whereas BoringSSL only allows the one with the time limitations.

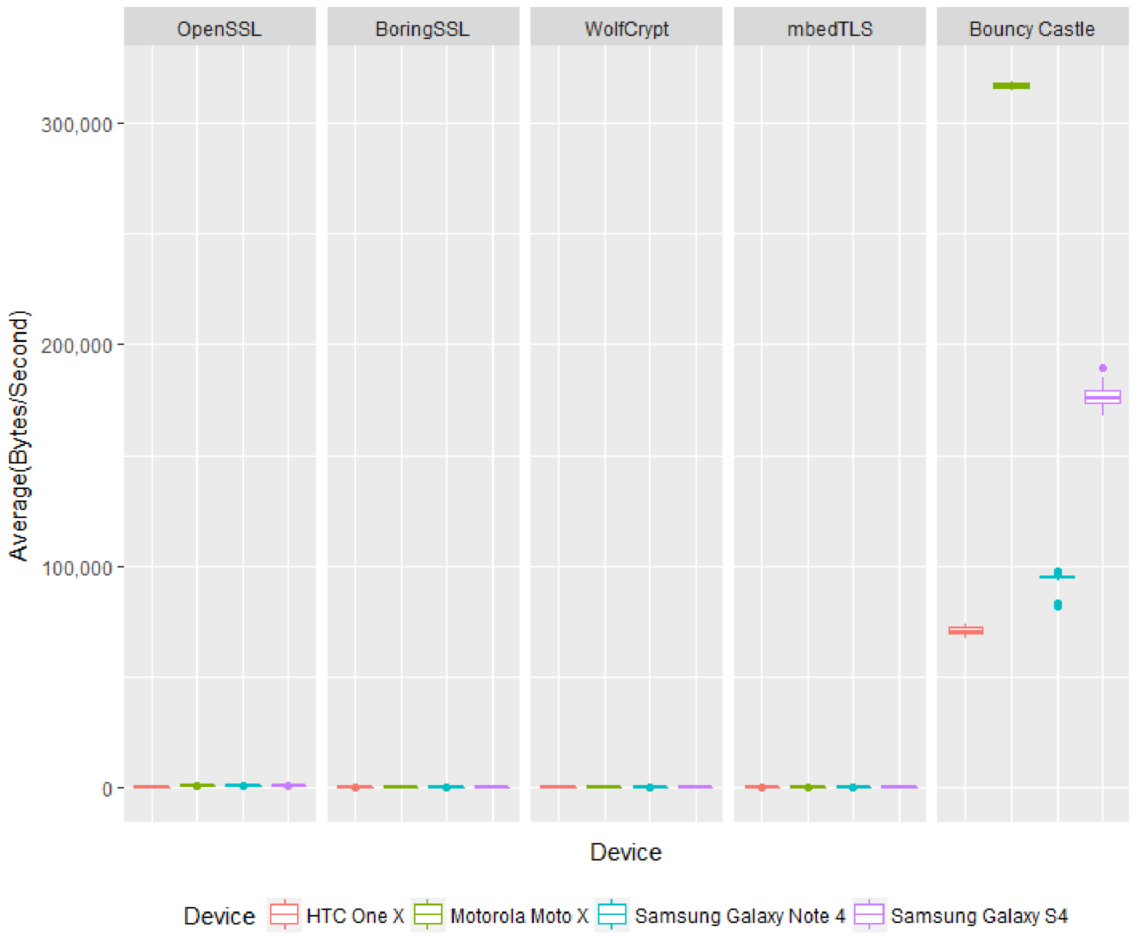


Figure 5.13: Results of ECDH Key agreement

The last benchmarks were done to check the Elliptic Curve Diffie-Hellman performance. First of all, as it happened in DH, the results of this algorithm can be slow because of the time constants that we previously mentioned. Surprisingly and for the first time, the best library (and the one a developer would have to implement) to perform this algorithm is Bouncy Castle. The results are clear and in this case, the Java security provider does not have any competitor. While OpenSSL can generate over 300 key agreements per seconds, Bouncy Castle can generate 300.000, a very big difference. As we can see, the results of ECDH are much slower than the results of Diffie-Hellman in the case of OpenSSL, but for the other libraries the key agreement/second rates are very consistent, Bouncy Castle being only the exception. What it is also surprising is the difference of results in the same libraries. For Bouncy Castle it is obvious that the Motorola device gets the better results with over 300.000 key agreements/second and the second best device (the Samsung Galaxy S4) will get results below 200.000 key agreements. By last, we can see that the differences between the HTC One X and the Samsung Galaxy Note 4 are very small. We can see that although for the other algorithms investigated the Note 4 was a very good device, we can not say the same for key agreement algorithms.

As we have seen in this chapter, OpenSSL is in the majority of the cases the best library to perform any cryptographic operation but if developers want to use only one specific algorithm in their app, they should be really aware of the library that they are using as this could result in a low performance of the device. We performed all the benchmarks under the same conditions but now we want to test if performing them at different times of the day means getting different or surprising results.

5.3 Time dependency of the algorithms/libraries

After making the performance tests, we wanted to see if trying the algorithms at different times of the day (morning,afternoon and night) the results and thus, the performance of them would vary. For these benchmarks we did not try every possible algorithm, we just selected what we considered more important to test, in this case: RSA, AES-CBC (1024 block size) and ECDH. Below are our results:

5.3.1 AES-CBC

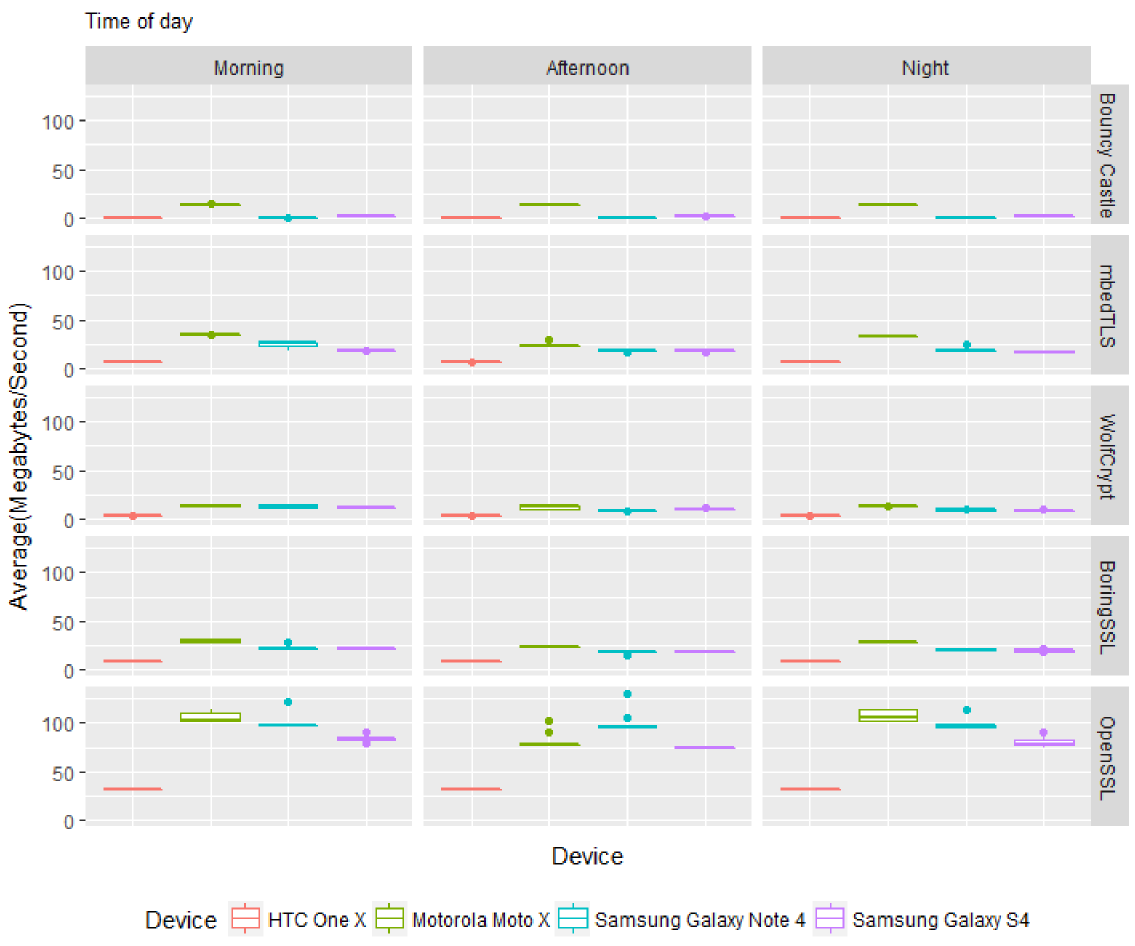


Figure 5.14: Results of the different libraries testing AES-CBC Encryption (1024 block size) in different times of the day

In this graphs we can see how the different libraries and devices behave in different moments of the day while encrypting a 1024 bytes string using AES-CBC as algorithm and operation mode. In this case, we will not talk about the differences in the libraries (as the code does not change and the results will be the same as in the "Performance tests") but we will talk about the differences between the devices. As we can see there is not clearly a significant difference of performance throughout the day, specially on the HTC One X and the Samsung Galaxy Note 4. On the contrary, we can see that the Motorola Moto X, although being one of the best device, shows more dispersion in the data when we encrypt in the morning and in the night, whereas the Samsung Galaxy Note 4, which also gets very good results, presents more consistent results during the different tests. Although it may be shocking that in this case, the Motorola device gets better results when in the performance test (5.2.1) the Samsung device got better results, the difference between the two devices is not so big and therefore we think it is not remarkable. We will not discuss the decryption of this algorithm because we have already seen that the decryption process is heavily correlated with the encryption, so the results will be very similar.

5.3.2 ECDH

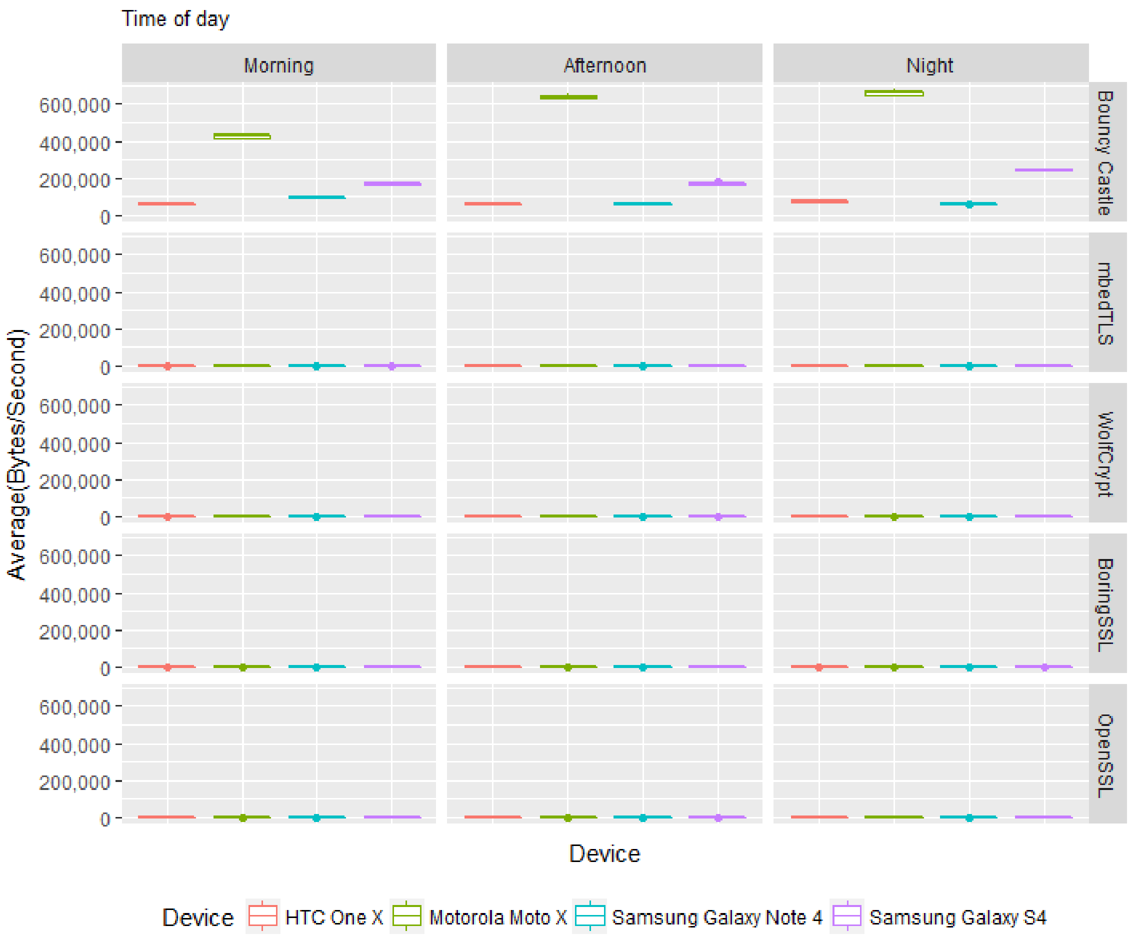


Figure 5.15: Results of the different libraries testing Elliptic Curve Diffie-Hellman in different times of the day

First of all, the most important aspect to comment is that the time of the day does not affect to the bad results of all the different libraries (except for Bouncy Castle) got. The results are pretty alike to the previous one and there is not any remarkable difference in the key agreements/second rate. What we can here see is that the performance of the algorithm is better at later hours, as we can clearly observe that the results are better at night. While in the morning, the Motorola Moto X can get over 400.000 key agreements/second, at night this rate ascends to over 600.000 key agreements. A very big difference. This results can also be observe in the Samsung Galaxy 4, as this will get better numbers when the night is approaching. On the contrary, we can see that the time of the day does not make any difference in the HTC One X and int the Samsung Galaxy Note 4, as the results they get are pretty much the same.

5.3.3 RSA

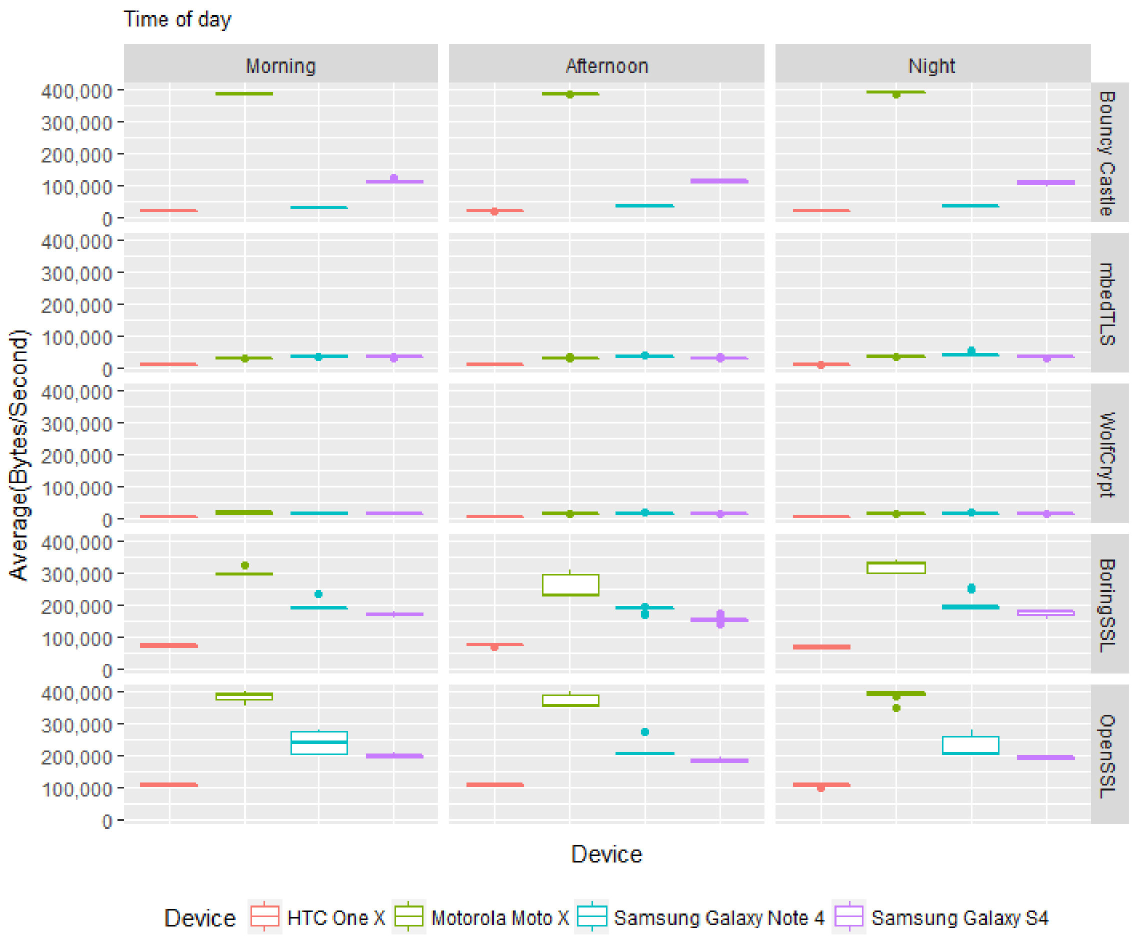


Figure 5.16: Results of the different libraries testing RSA encryption in different times of the day

For RSA we will discuss the encryption and the decryption process, because the differences seen in 5.2.5 are so significant that it is necessary to analyze both processes. We can clearly see that the results in the different times of the day are pretty solid and there are not big differences, specially in Bouncy Castle, mbedTLS and wolfCrypt, where there is not any variance between the results. We can see that in BoringSSL and in OpenSSL the Motorola Moto X and the Samsung Galaxy Note 4 do present more variance than the other devices, but this happens in almost every algorithm researched, so this must be not a problem of the different libraries but from the different CPUs. The only difference that we can really appreciate between the different times of the day is that, when using OpenSSL and BoringSSL, the Motorola Moto X will present a really small decrease of its activity on the afternoon. This decrease will be more clearly to see on the decryption process.

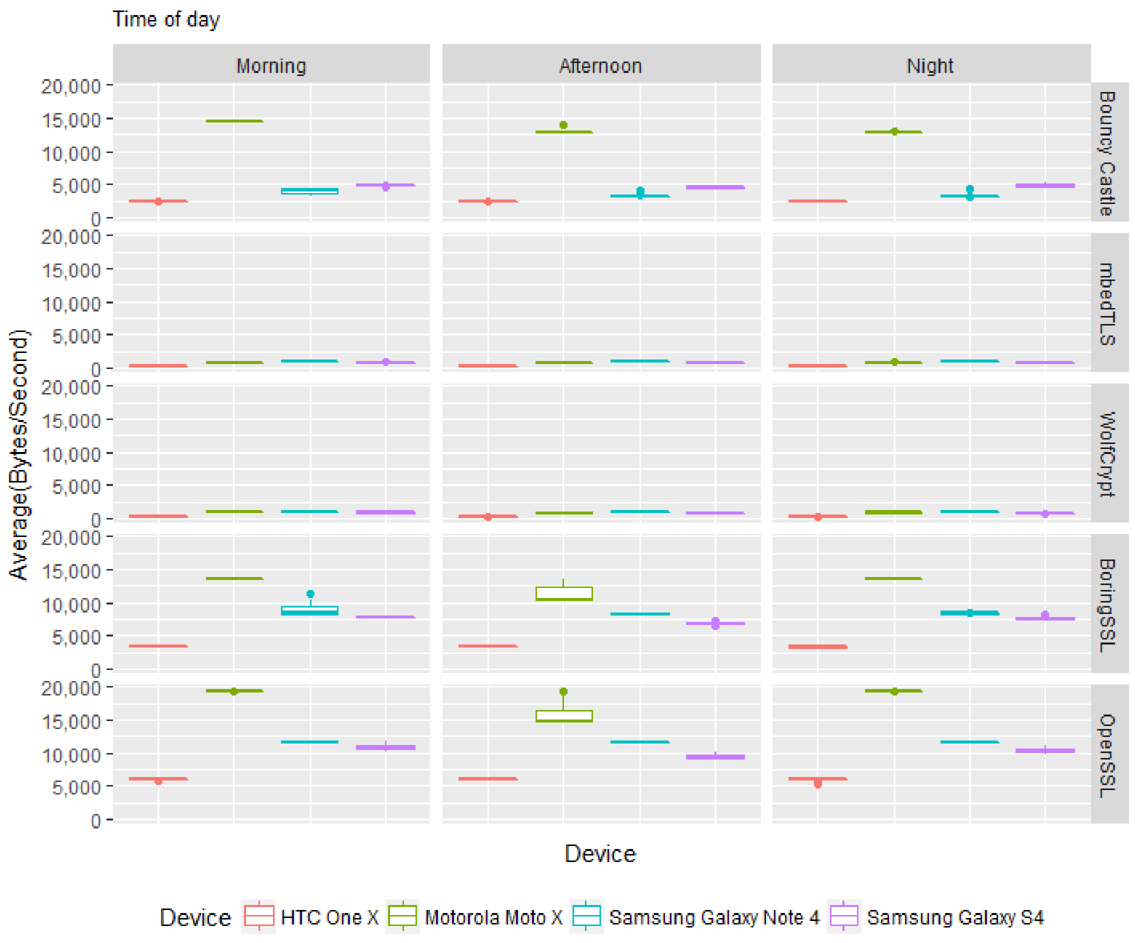


Figure 5.17: Results of the different libraries testing RSA decryption in different times of the day

As we can see, there are few differences in the library between the different times of the day. Whereas in mbedTLS and wolfCrypt the changes are inappreciable, in Bouncy Castle, BoringSSL and OpenSSL there are changes. These changes are always related to the Motorola Moto X, which is the most unstable device. While using Bouncy Castle, the mobile phone will present better results in the mornings, on the contrary, while using BoringSSL and OpenSSL, the device will present worse results on the afternoon. Apart from these differences, we do not see any significant difference that is remarkable to comment.

Having seen the results of the different devices and libraries throughout the day, now we would like to comment how do the temperature of the device varies throughout the different processes.

5.4 Temperature test

Analyzing the temperature of the CPU of the devices while performing the different algorithms is important for the developer, as it can happen, that although a library can be very efficient while performing one of the cryptographic operations, the temperature of the device will increase in a way that will affect the life of the CPU and, therefore, the life of the device. As developers do not want that their apps produce in such a catastrophe, they should also be aware of this data. Before explaining any of the results that we got, the first and one of the most important conclusion that we achieved is that the temperature of the device does not influence the performance. As the performance results will often present this big variance between the data, a peak in the temperature will not result in a better MBytes/second rate.

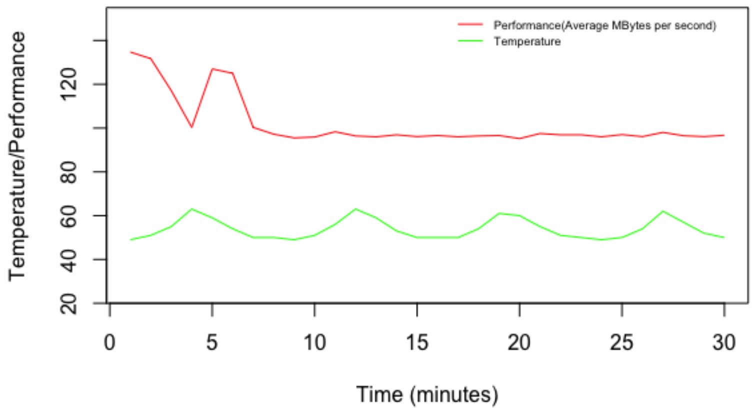


Figure 5.18: Comparison of the performance (Average Mbytes) rate and the temperature in a 30 minutes encryption using AES-CBC (OpenSSL) with a 1024 bytes string in the Samsung Galaxy Note 4.

As we can see in the previous figure, the performance of the device is not related to the temperature as the algorithm will present a better performance at the beginning, when the temperature is not so high and then the performance will stabilize whereas the temperature will be varying throughout the whole process.

Due to time limitations, we will not analyze every single library, but we will select what we consider most important to comment. The results are presented below:

5.4.1 AES-CBC

As OpenSSL has the best results, we consider that a developer would always want to use AES-CBC for their encryption/decryption operations. For the encryption, there are a few things that are necessary to comment. First of all, we will in all these temperature tests see how the Motorola Moto X has the most solid CPU: the Qualcomm Snapdragon 808. Throughout the tests, in very few cases the temperature of the CPU will go over 35 °C. This phone also has the lower initial temperature: Whereas the other CPUs are usually around 40 °C, the

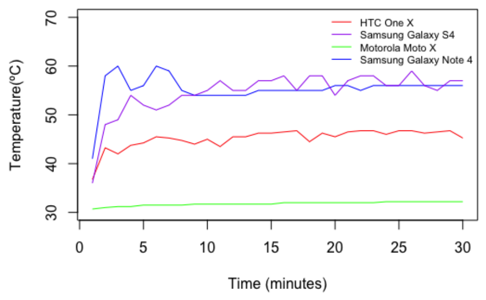


Figure 5.19: Results of performing AES-CBC Encryption over 1024 bytes strings using OpenSSL as library (less is better)

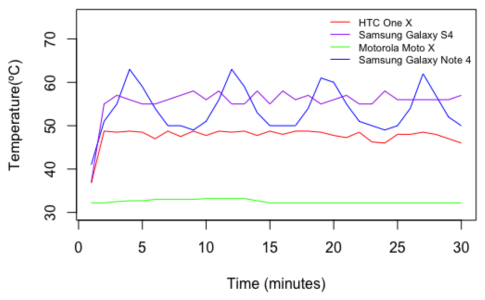


Figure 5.20: Results of AES-CBC Decryption performing OpenSSL over 1024 bytes strings using OpenSSL as library (less is better)

Snapdragon 808 is around 32 °C. We can also see that the HTC One, although it has not a very good performance, it does not present a big differences through the whole encryption process. On the other hand, the Samsung Galaxy S4 does present big changes and one of the highest temperature. Although the device starts at around 38°C, the phone will reach around 58 °C, which means a rise of 20°C. The Samsung device also presents very unstable results, because it varies through 55°C and 58 °C through the whole process and it never gets stabilized. The Samsung Galaxy Note, which in all the temperature tests done is the most unstable device, here it presents a big rise of the temperature the first 7–8 minutes of encrypting but then it will stabilize itself at around 56 °C.

For the decryption process, the devices will present the same behavior but with a rise of a few grads in every result. The Motorola will start at 33°C and then it will remain around this temperature through the whole process. The HTC device and the Samsung Galaxy will have the same behavior of rise-then-stabilize that they had in the encryption process, but this time the temperature will rise a little bit in the case of the HTC One (around 48 °C). Ultimately we have the Samsung Galaxy Note. As we can clearly see, this processor (Qualcomm Snapdragon 805) presents a very unstable temperature. The temperature is constantly varying in cycles, which starts at around 50°C and finishes at over 60 °C. Taking into account that the original temperature of the device is around 40 °C, this means a rise of around 23 °C. Really a big difference that developers should keep in mind.

5.4.2 AES-CTR

For AES-CTR we will also use the library with the best results, in this case, mbedTLS. First of all, this will be one of the few cases where the Motorola Moto X will present temperatures higher than 40 °C. In this case, we can see that although the device has higher temperatures, its stability remains untouched, as there are not big changes during the process. The temperature of the device will later be reduced during the decryption process, reaching more "normal" temperatures, around 33 °C. Surprising is, that with this library we can observe how the HTC One X has a very similar performance than the Motorola Moto X. During the encryption process, the differences will be very small (around 2 degrees). On the decryption process, the HTC device will still present the same stability, having temperatures around 45 °C. So

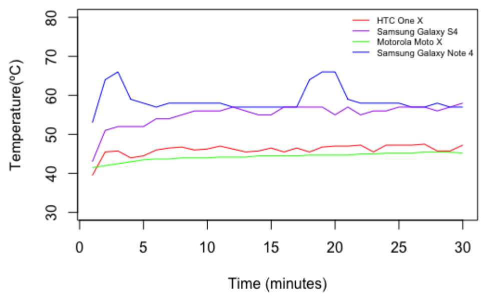


Figure 5.21: Results of performing AES-CTR encryption using over 1024 bytes strings using mbedTLS as library (less is better)

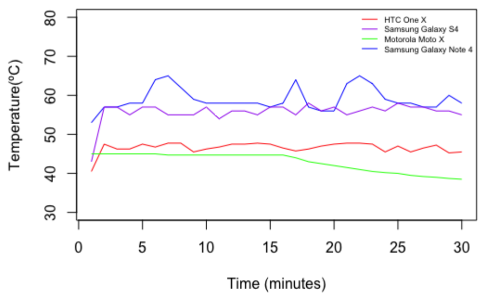


Figure 5.22: Results of performing AES-CTR decryption using over 1024 bytes strings using mbedTLS as library (less is better)

we could say that this device presents the highest stability between the encryption/decryption processes. The Samsung Galaxy S4 also has this stability, but as with AES-CBC, the temperatures are much higher: around 55°C in both processes. In this algorithm we can also see the instability of the Samsung Galaxy Note 4. For this algorithm, the device presents a little better than in AES-CBC, but it still presents big changes. In this case, we will not be speaking about cycles, but we will talk about peaks that repeat in random periods, reaching temperatures around 66°C. This CPU, as we can see, overheats a lot.

5.4.3 AES-GCM

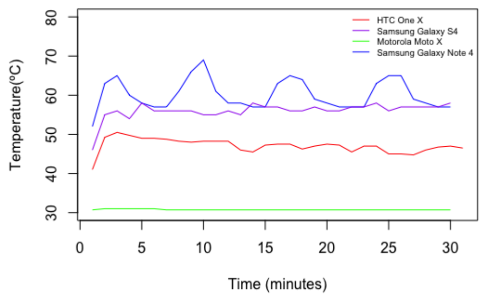


Figure 5.23: Results of performing AES-GCM encryption using over 1024 bytes strings using mbedTLS as library (less is better)

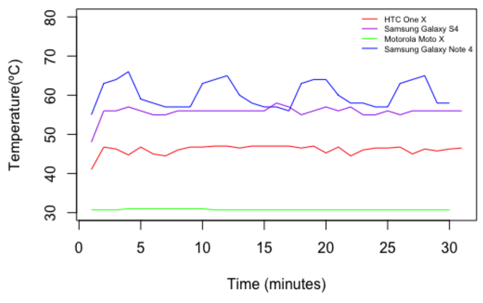


Figure 5.24: Results of performing AES-GCM DEcryption using over 1024 bytes strings using mbedTLS as library (less is better)

In the AES-GCM we will discuss again the mbedTLS. This algorithm presents the same patterns as AES-CBC. The Motorola Moto X will have the best performance, around 30 °C. The HTC One X will have again a very good stability but nevertheless will still have higher temperatures than the Motorola device. Again, the Samsung Galaxy Note 4 will present this cyclical pattern and this time it will reach temperatures around 70 °C, one of the highest temperatures recorded. By last, the Samsung Galaxy S4 will reach high temperatures of around 55 °C but these ones will remain stable during both encrypting and decrypting process.

5.4.4 AES-OFB

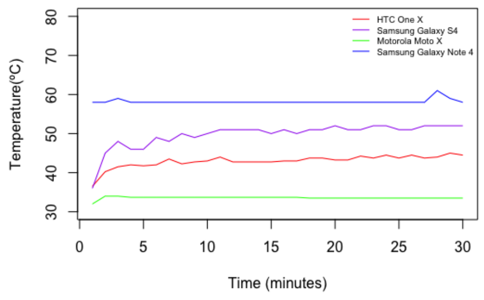


Figure 5.25: Results of performing AES-OFB encryption using over 1024 bytes strings using Bouncy Castle as library (less is better)

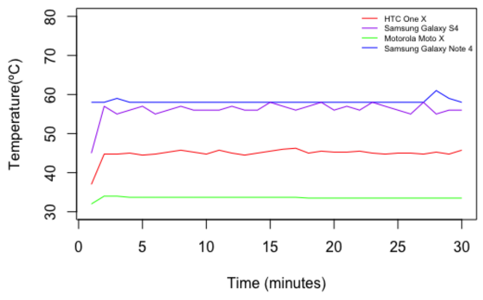


Figure 5.26: Results of performing AES-OFB decryption using over 1024 bytes strings using Bouncy Castle as library (less is better)

The performance of this algorithms is clearly really good. As we can see, there are not big changes throughout the processes and every device remains stable. This is not new for devices such as the Motorola Moto X or the HTC One, whose results are pretty similar to the others researched, but for both Samsung devices it is the first time we see this. As we can appreciate, the Samsung Galaxy Note 4 remains with an almost perfect stability during the both process with the exception of some small peaks. The Samsung Galaxy S4 on the contrary, will present more small temperatures changes. In the encryption process, the temperature of the device will slowly rise from around 40 °C to over 50 °C, where the CPU will stabilize itself. The decryption process will present this rise in a faster way but then, it will stabilize at the same temperature as in the encryption process.

5.4.5 Diffie-Hellman

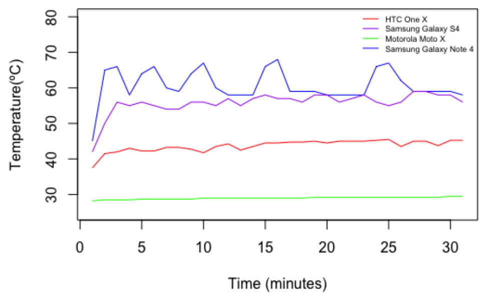


Figure 5.27: Results of generating key agreements with Diffie-Hellman using mbedTLS as library (less is better)

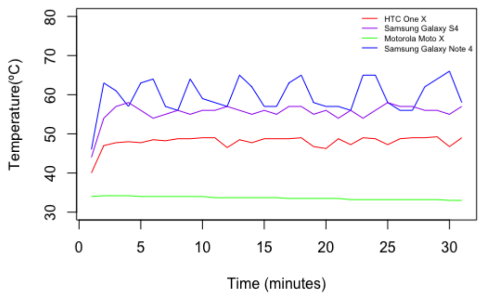


Figure 5.28: Results of generating key agreements with Diffie-Hellman using Bouncy Castle as library (less is better)

For analyzing the results of Diffie-Hellman we did not only choose the best library (OpenSSL) but we also took the sample of mbedTLS. As we saw in 5.2.7, the results of mbedTLS were very bad and we also wanted to research, if apart from generating key agreements in such a slow rate, the temperature is affected in the same way or on the contrary, it will not make use of the whole power of the CPU. As we can see, it is surprising how both libraries present almost the same results. First of all, we can see one of the lowest temperature of the Motorola Moto X. Surprisingly, in spite of its bad results, the Motorola device will have a temperature of around 30 °C using mbedTLS whereas using OpenSSL, the temperature rises around 35 °C. Logically we can understand these results seeing how well OpenSSL performs. The HTC One X will also have a similar performance. This time, the HTC device will also present slightly higher temperatures using OpenSSL than using mbedTLS. Nevertheless, both libraries will be in a range of 40–50 °C. Ultimately, both of the Samsung devices present the same performance while using the two libraries. On the one hand, we have the Samsung Galaxy Note 4, which as always will present this cyclical pattern. On the other hand we have the Samsung Galaxy Note 4, which will present this unstable temperature but in a smaller range (around 55°C).

5.4.6 Elliptic Curve Diffie Hellman

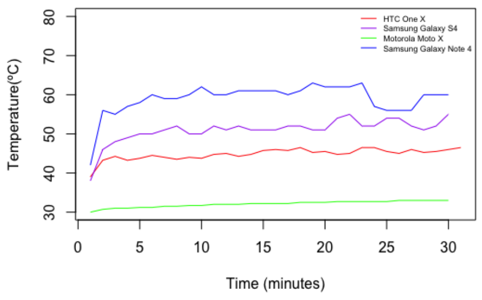


Figure 5.29: Results of generating key agreements with Elliptic Curve Diffie-Hellman using mbedTLS as library (less is better)

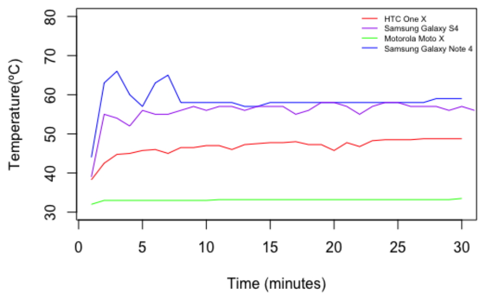


Figure 5.30: Results of generating key agreements with Elliptic Curve Diffie-Hellman using Bouncy Castle as library (less is better)

The temperature performance that we will investigate in this case will be Bouncy Castle and wolfCrypt. We selected the Java library because it presents the best results while using ECDH and we chose wolfCrypt because it presents as bad results as mbedTLS while generating a key agreement. Surprisingly, the performance of the devices while using ECDH and these libraries is surprisingly very good. The Motorola Moto X, like in almost every algorithm, behaves in the same way and remains stable in range of 30–33°C. Then, most surprising to see are the other devices. First of all, we can see that the Samsung Galaxy Note, although it still is unstable, it does not present anymore this cyclical pattern. The temperature will be very high at the beginning while using wolfCrypt (around 67°C), but then it will get stabilized at 60°C. On the contrary, while using Bouncy Castle the device's CPU will be more unstable than with wolfCrypt, it remains between 55-60°C. The Samsung Galaxy S4 will have a similar behavior than the Note 4 but in a colder range. While using wolfCrypt will be around 60°C (it has a very similar performance than the Samsung Galaxy Note 4) and will be around 50°C while using Bouncy Castle. The HTC One X in this case will have the same behavior in both libraries, it will stay around 43°C.

5.4.7 MD-5

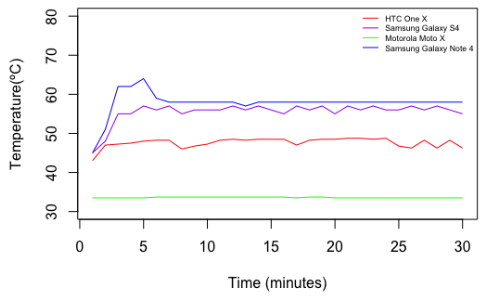


Figure 5.31: Results of hashing 1024 bytes strings using OpenSSL as library (less is better)

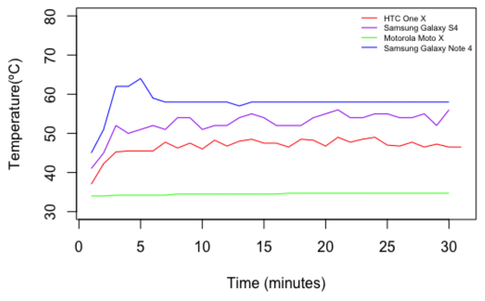


Figure 5.32: Results of hashing 1024 bytes strings using OpenSSL as library (less is better)

To analyze the heating performance while hashing with MD-5 we selected OpenSSL, which presented the best results and also Bouncy Castle, because as we could see in 5.2.6, the Motorola Moto X presented a big dispersion between the data and we would like to see if this variance is also correlated with the temperature the device can achieve. Surprisingly, we can see that this is not the case. Both of the graphs seems exactly the same and although the HTC One X and the Samsung Galaxy S4 present a bit more unstable results, there are not big differences. The Motorola Moto X has an average of 35ºC while hashing with both libraries, the HTC One X around 45ºC, the Samsung Galaxy S4 around 55ºC and the Samsung Galaxy Note 4 around 58%. This behavior will also repeat with the remaining 3 libraries, so a developer who just wants to hash data using MD-5 could choose directly one of the best performing libraries (OpenSSL or Bouncy Castle) as the temperature of the device will not change drastically.

5.4.8 RSA

Due to the performance results of the RSA algorithm that the we achieved in 5.2.5, we considered interesting to see the performance of the different devices on 3 different libraries: OpenSSL, Bouncy Castle and BoringSSL. Obviously we will research not only the encryption but also the decryption process. As we could see, the decryption process was much slower than the encryption, so we supposed that the CPU will overheat more with this process, let's see if this later becomes true.

First of all, we can see that the results of the different devices remain constant, but as we had previously supposed, the decryption process is a bit more demanding than the encryption process. There are not big changes and the temperature is pretty alike to the other algorithms. For the encryption process, the Motorola Moto X is the best device getting as usual an average temperature of 32ºC for the encryption and around 35ºC for the decryption. Then, the second device with the best mean temperature is as always the HTC One X. This phone has a temperature of around 45ºC during the encryption and 49ºC during the decryption. At last we have the two Samsung devices. It is unexpected that the Note 4, being the RSA such a demanding algorithm, does not reach very high temperatures. As for both processes

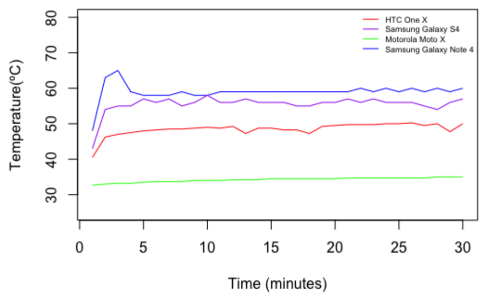


Figure 5.33: Temperature results of a continuous encryption with RSA using OpenSSL as library (less is better)

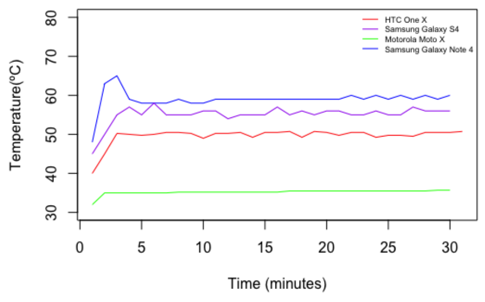


Figure 5.34: Temperature results of a continuous decryption with RSA using OpenSSL as library (less is better)

it presents almost the same performance with an average temperature of 59°C. The Samsung Galaxy S4 also gets this results but instead if a mean of 59°C, it is of 57°C, a really small difference taking into account the big differences in the byte/second rate.

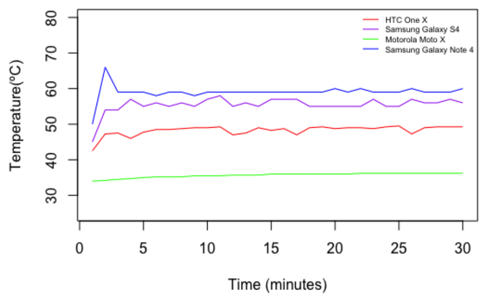


Figure 5.35: Temperature results of a continuous encryption with RSA using BoringSSL as library (less is better)

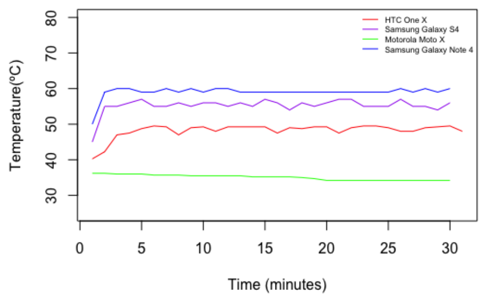


Figure 5.36: Temperature results of a continuous decryption with RSA using BoringSSL as library (less is better)

The results of BoringSSL are very similar to the results of OpenSSL. Although this shouldn't be a big surprise as they share code[2], they do not present the similar results in the bytes/second rate. Having this similar results, a developer who just wants the best performance, should go straight for using OpenSSL, as it wouldn't make any sense using the Google's library. The most significant aspects to comment is that the Samsung Galaxy Note 4 does not present at the beginning of the decryption process this characteristic peak and instead, it will start with a really stabilized temperature. Also, as we can see, the Motorola Moto X starts with a really high temperature during the decryption process and then it will get stable. As we have previously seen, the Motorola device always present a strong and stable CPU, so we will not note this high temperature as a bad results as then it will get lower.

To finish our temperature performance analysis, we will comment the results of Bouncy Castle. Although the results of this library were not as good as the ones from BoringSSL and OpenSSL, we wanted to see if the performance of the CPU while using Java code was better than with the other libraries. For this case we will not comment the results of the Motorola Moto X and the Samsung Galaxy S4, as they are like the previously ones. First of all, we can appreciate that the encrypting process in the Samsung Galaxy Note 4 is very

[2]Reminder: BoringSSL is a fork of OpenSSL and thus, they share the majority of the code

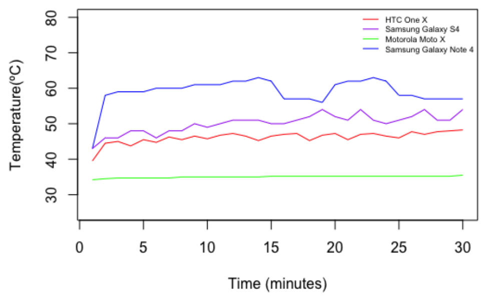


Figure 5.37: Temperature results of a continuous encryption with RSA using BoringSSL as library (less is better)

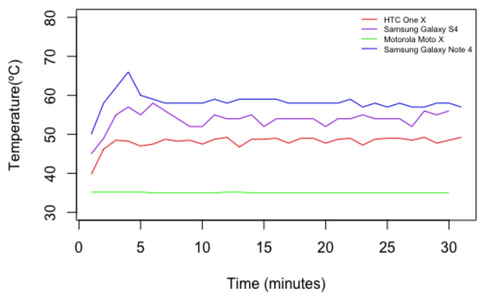


Figure 5.38: Temperature results of a continuous decryption with RSA using BoringSSL as library (less is better)

unstable. We can not here see the pattern that we saw in algorithms such as AES, but we see how the temperatures fluctuate between 55°C and 63°C, very big differences having into account that this didn't happen in the other libraries. Also it is surprising how unstable the Samsung Galaxy S4 behaves, as it shows a continuously rising of the temperatures until it get stabilized at around 55°C. On the contrary, in the decryption process both devices will have a more stabilized behaviour and the temperatures will not fluctuate so much.

We have already seen how the different cryptographic libraries behave in different environments, measuring their encryption (and/or decryption)/second rate or key agreement/second rate, seeing if the results change in dependency with the moment of the day and seeing how the temperature of the devices changes during the different process.

6 Conclusion

After seeing how the different devices and libraries behave we are now able to choose and recommend which library and/or device would work better under very different circumstances. First of all, we have to take into account that every situation is different and although the results of one of the libraries might be better, sometimes it would make sense to use another library. We have always spoken about the numeric data from our research to make our conclusions and there are some aspects that we do not have in mind. In the first place, we are assuming that the difficulty of implementing every library is the same. Although veteran developers will not surely have any problems implementing them, unexperienced developers will have problems implementing some of the libraries as sometimes the documentation lacks of details and ways to implement them in an optimized way. Also we have to take into account that the day-lasting tests were done in summer, as we can see, the good-performance of some of the devices by night might be because the ambient temperature of the room has decreased.

Regarding the temperature tests we were just measuring the CPU temperature. Despite the fact that the CPU will be doing all the work while performing the different operations, the temperature of the device will not be the same as the processor's temperature. The construction materials and the battery temperature play also a role in the final temperature and this can change the user's experience. As we have seen, the Motorola Moto X is the device that presents the better CPU temperatures, this should mean that the device should be colder than the others but it is not. While performing the different operations, the temperature of the device was so high that it made difficult to hold it in the hand. Also the size of every library is important. Developers should also have in mind the size of their apps and in some cases, for example developing an Android Go app where the app must be smaller than 40MB[autl], consuming as few resources as possible, it is not only very important but a must. Of course, the size of every library can be shrunk by just taking only the files needed, but this would mean more work time and in some cases, this resource is also limited.

Although we are pushing the devices to their boundaries with the different benchmarks, these are some of the facts that developers have to value making decision.

Now, going straight to the results, if a developer would have to use all these algorithms and he just could implement one of the libraries, he should go straight for OpenSSL. In only few cases OpenSSL is not the best library for every device and algorithm. If it is not the best, it will always present one of the best bytes(and/or key agreements)/second rate. Besides it also presents a very good CPU-heating average between the different devices. This library is the best for performing AES-CBC, AES-OFB, RSA, MD-5 and Diffie-Hellman operations, over 60% of the algorithms researched. Only in the cases of Elliptic Curve Diffie Hellman (where we would use Bouncy Castle), AES-CTR and AES-GCM (where we would implement mbedTLS) is not a good idea not to use OpenSSL.

Regarding the devices, we have seen that the Qualcomm Snapdragon 808 CPU used in the Motorola Moto X is the one who presents the best results. It is a very stable processor and the results are always below 35°C, which is a very good results having in mind that the normal range of a mobile phone's CPU is in a range between 37–43°C [Cor18]. The older version of the Qualcomm Snapdragon 808, the Snapdragon 805 (implemented in the Samsung Galaxy Note 4) is much more unstable and performing different cryptographic operations during a long periods of time will result in a shorter device's life. The NVIDIA Tegra 3, which is on the HTC One X, is also very unstable but this fact is not surprising as it is a much cheaper processor.

With all these facts now we should be able to implement different cryptographic operations knowing which one will perform better and not harm the CPU life.

7 Appendix

The results of the different tests performed in 5.2 are presented in this section in a more extended and exact way.

7.1 C-Transfer Results

Device	Size of String	Mean (nanoseconds)
Motorola Moto X	128	6105
Motorola Moto X	256	11494
Motorola Moto X	512	17451
Motorola Moto X	1024	32620
HTC One X	128	11943
HTC One X	256	19676
HTC One X	512	21838
HTC One X	1024	55662
Samsung Galaxy S4	128	18889
Samsung Galaxy S4	256	24483
Samsung Galaxy S4	512	35145
Samsung Galaxy S4	1024	55827
Samsung Galaxy Note 4	128	17201
Samsung Galaxy Note 4	256	19655
Samsung Galaxy Note 4	512	24931
Samsung Galaxy Note 4	1024	36511

Table 7.1: This table shows the average time that the different devices need to handle native code

7.2 AES-CBC Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	8.5	8.7	8.7
HTC One X	256	8.6	8.7	8.9
HTC One X	512	8.6	8.8	9.0
HTC One X	1024	8.7	8.8	9.0
Samsung Galaxy S4	128	18.9	20.6	22.5
Samsung Galaxy S4	256	17.4	19.8	21.6
Samsung Galaxy S4	512	17.5	19.2	22.4
Samsung Galaxy S4	1024	18.7	20.0	23.0
Samsung Galaxy Note 4	128	12.5	21.0	29.3
Samsung Galaxy Note 4	256	12.2	20.1	29.3
Samsung Galaxy Note 4	512	12.3	16.8	27.5
Samsung Galaxy Note 4	1024	12.6	17.7	29.4
Motorola Moto X	128	10.1	10.3	10.5
Motorola Moto X	256	10.4	10.4	10.4
Motorola Moto X	512	10.4	10.4	10.4
Motorola Moto X	1024	10.3	10.3	10.4

Table 7.2: AES-CBC Boring SSL Encryption

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	8.0	8.2	8.3
HTC One X	256	8.0	8.2	8.4
HTC One X	512	8.1	8.3	8.4
HTC One X	1024	8.2	8.4	8.4
Samsung Galaxy S4	128	18.2	19.5	20.7
Samsung Galaxy S4	256	16.8	17.6	18.5
Samsung Galaxy S4	512	18.4	19.4	21.1
Samsung Galaxy S4	1024	18.2	19.1	20.7
Samsung Galaxy Note 4	128	12.5	16.9	27.3
Samsung Galaxy Note 4	256	12.9	17.5	27.4
Samsung Galaxy Note 4	512	12.0	15.9	25.6
Samsung Galaxy Note 4	1024	12.5	17.0	27.7
Motorola Moto X	128	10.1	10.2	10.2
Motorola Moto X	256	10.3	10.3	10.3
Motorola Moto X	512	10.3	10.3	10.3
Motorola Moto X	1024	10.2	10.3	10.4

Table 7.3: AES-CBC Boring SSL Decryption

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	30.2	30.8	31.1
HTC One X	256	30.5	31.2	31.7
HTC One X	512	30.8	31.4	31.7
HTC One X	1024	30.9	31.5	32.1
Samsung Galaxy S4	128	72.1	80.0	94.1
Samsung Galaxy S4	256	68.9	74.9	86.3
Samsung Galaxy S4	512	58.8	72.6	93.0
Samsung Galaxy S4	1024	75.0	83.0	94.3
Samsung Galaxy Note 4	128	51.1	83.5	123.3
Samsung Galaxy Note 4	256	51.5	76.9	121.9
Samsung Galaxy Note 4	512	55.0	79.5	129.7
Samsung Galaxy Note 4	1024	57.4	93.8	134.7
Motorola Moto X	128	59.8	66.9	102.3
Motorola Moto X	256	60.9	10.0	112.2
Motorola Moto X	512	61.5	63.8	102.4
Motorola Moto X	1024	61.3	61.9	63.0

Table 7.4: AES-CBC OpenSSL Encryption

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	29.8	30.7	31.0
HTC One X	256	30.5	31.3	31.6
HTC One X	512	30.8	31.5	31.7
HTC One X	1024	30.8	31.5	31.8
Samsung Galaxy S4	128	69.0	72.1	77.4
Samsung Galaxy S4	256	69.8	77.0	88.0
Samsung Galaxy S4	512	59.1	66.1	71.2
Samsung Galaxy S4	1024	75.6	82.3	91.5
Samsung Galaxy Note 4	128	52.5	76.8	118.4
Samsung Galaxy Note 4	256	52.1	76.5	119.2
Samsung Galaxy Note 4	512	51.6	65.2	107.5
Samsung Galaxy Note 4	1024	54.7	78.3	121.3
Motorola Moto X	128	57.4	57.6	59.7
Motorola Moto X	256	58.0	58.5	58.5
Motorola Moto X	512	57.9	59.1	61.1
Motorola Moto X	1024	59.1	59.5	61.5

Table 7.5: AES-CBC OpenSSL Decryption

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	2.4	2.8	3.0
HTC One X	256	2.5	2.8	3.3
HTC One X	512	2.6	2.7	2.9
HTC One X	1024	3.5	3.5	3.6
Samsung Galaxy S4	128	7.0	7.7	8.2
Samsung Galaxy S4	256	8.1	8.8	9.5
Samsung Galaxy S4	512	8.6	9.3	11.4
Samsung Galaxy S4	1024	9.0	10.3	11.2
Samsung Galaxy Note 4	128	4.9	5.5	6.0
Samsung Galaxy Note 4	256	6.2	6.8	7.9
Samsung Galaxy Note 4	512	5.7	6.1	9.5
Samsung Galaxy Note 4	1024	6.0	6.6	12.4
Motorola Moto X	128	3.4	3.4	3.4
Motorola Moto X	256	4.4	4.4	4.7
Motorola Moto X	512	5.3	5.3	5.4
Motorola Moto X	1024	6.2	6.2	6.2

Table 7.6: AES-CBC wolfCrypt Encryption

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	2.2	2.5	2.7
HTC One X	256	2.2	2.4	2.7
HTC One X	512	2.4	2.5	2.7
HTC One X	1024	3.1	3.2	3.2
Samsung Galaxy S4	128	6.8	7.1	7.5
Samsung Galaxy S4	256	7.7	8.2	8.7
Samsung Galaxy S4	512	8.3	8.6	9.2
Samsung Galaxy S4	1024	8.5	9.2	10.0
Samsung Galaxy Note 4	128	4.6	4.9	5.1
Samsung Galaxy Note 4	256	5.5	6.2	6.5
Samsung Galaxy Note 4	512	5.2	5.6	5.8
Samsung Galaxy Note 4	1024	5.8	6.2	6.4
Motorola Moto X	128	3.3	3.3	3.3
Motorola Moto X	256	4.3	4.3	4.3
Motorola Moto X	512	5.2	5.2	5.2
Motorola Moto X	1024	6.1	6.1	6.1

Table 7.7: AES-CBC OpenSSL Decryption

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	6.3	6.4	6.5
HTC One X	256	6.3	6.4	6.5
HTC One X	512	6.2	6.5	6.5
HTC One X	1024	6.3	6.5	6.5
Samsung Galaxy S4	128	17.1	18.3	19.2
Samsung Galaxy S4	256	17.2	18.2	19.2
Samsung Galaxy S4	512	16.7	17.8	19.5
Samsung Galaxy S4	1024	15.7	16.9	19.1
Samsung Galaxy Note 4	128	18.8	22.5	26.5
Samsung Galaxy Note 4	256	11.3	17.9	26.0
Samsung Galaxy Note 4	512	11.3	17.2	25.7
Samsung Galaxy Note 4	1024	10.9	15.4	24.4
Motorola Moto X	128	11.3	11.3	11.4
Motorola Moto X	256	11.2	11.5	15.4
Motorola Moto X	512	11.3	11.4	11.4
Motorola Moto X	1024	11.4	13.2	32.1

Table 7.8: AES-CBC mbedTLS Encryption

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	6.5	6.7	6.7
HTC One X	256	6.5	6.7	7.6
HTC One X	512	6.5	6.7	6.8
HTC One X	1024	6.6	6.7	6.8
Samsung Galaxy S4	128	16.7	17.8	18.6
Samsung Galaxy S4	256	15.8	17.8	19.0
Samsung Galaxy S4	512	16.1	17.3	18.5
Samsung Galaxy S4	1024	15.2	16.2	17.5
Samsung Galaxy Note 4	128	17.7	18.7	19.0
Samsung Galaxy Note 4	256	9.3	15.6	24.8
Samsung Galaxy Note 4	512	10.1	15.5	24.8
Samsung Galaxy Note 4	1024	10.5	14.3	23.5
Motorola Moto X	128	11.6	11.6	11.6
Motorola Moto X	256	11.6	11.9	20.8
Motorola Moto X	512	11.6	11.7	11.7
Motorola Moto X	1024	11.7	11.7	11.7

Table 7.9: AES-CBC mbedTLS Decryption

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	0.1	0.1	0.1
HTC One X	256	0.1	0.1	0.1
HTC One X	512	0.1	0.1	0.1
HTC One X	1024	0.1	0.1	0.1
Samsung Galaxy S4	128	1.4	1.5	1.7
Samsung Galaxy S4	256	1.5	1.6	1.7
Samsung Galaxy S4	512	1.6	1.7	1.9
Samsung Galaxy S4	1024	1.6	1.7	1.9
Samsung Galaxy Note 4	128	0.1	0.1	0.1
Samsung Galaxy Note 4	256	0.1	0.1	0.1
Samsung Galaxy Note 4	512	0.1	0.1	0.1
Samsung Galaxy Note 4	1024	0.1	0.1	0.1
Motorola Moto X	128	4.6	4.6	4.7
Motorola Moto X	256	5.0	5.0	5.1
Motorola Moto X	512	5.2	5.4	8.9
Motorola Moto X	1024	5.4	5.4	5.5

Table 7.10: AES-CBC Bouncy Castle Encryption

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	0.1	0.1	0.1
HTC One X	256	0.1	0.1	0.1
HTC One X	512	0.1	0.1	0.1
HTC One X	1024	0.1	0.1	0.1
Samsung Galaxy S4	128	1.4	1.5	1.6
Samsung Galaxy S4	256	1.4	1.5	1.6
Samsung Galaxy S4	512	1.6	1.6	1.8
Samsung Galaxy S4	1024	1.6	1.7	1.7
Samsung Galaxy Note 4	128	0.1	0.1	0.1
Samsung Galaxy Note 4	256	0.1	0.1	0.1
Samsung Galaxy Note 4	512	0.1	0.1	0.1
Samsung Galaxy Note 4	1024	0.1	0.1	0.1
Motorola Moto X	128	4.6	4.6	4.7
Motorola Moto X	256	4.9	4.9	4.9
Motorola Moto X	512	5.2	5.2	5.2
Motorola Moto X	1024	5.4	5.4	5.4

Table 7.11: AES-CBC Bouncy Castle Decryption

7.3 AES-CTR Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	27.9	28.5	28.9
HTC One X	256	28.3	28.9	29.4
HTC One X	512	28.6	29.3	29.8
HTC One X	1024	28.6	29.4	29.8
Samsung Galaxy S4	128	38.3	42.9	47.7
Samsung Galaxy S4	256	60.8	65.9	75.6
Samsung Galaxy S4	512	61.3	64.9	77.3
Samsung Galaxy S4	1024	63.1	66.4	78.5
Samsung Galaxy Note 4	128	76.4	86.7	107.7
Samsung Galaxy Note 4	256	77.8	79.6	93.7
Samsung Galaxy Note 4	512	82.1	85.1	98.7
Samsung Galaxy Note 4	1024	82.8	85.1	11.0
Motorola Moto X	128	54.7	54.8	54.9
Motorola Moto X	256	55.8	55.8	56.0
Motorola Moto X	512	56.2	56.5	57.3
Motorola Moto X	1024	56.8	56.9	56.9

Table 7.12: AES-CTR OpenSSL Encryption Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	27.7	28.2	28.7
HTC One X	256	27.2	28.8	29.3
HTC One X	512	28.6	29.2	29.7
HTC One X	1024	27.9	29.4	31.5
Samsung Galaxy S4	128	35.4	37.9	42.5
Samsung Galaxy S4	256	58.4	60.7	63.6
Samsung Galaxy S4	512	60.8	62.4	64.7
Samsung Galaxy S4	1024	63.8	65.0	65.9
Samsung Galaxy Note 4	128	73.4	76.4	78.4
Samsung Galaxy Note 4	256	66.9	73.2	79.7
Samsung Galaxy Note 4	512	80.9	83.2	85.4
Samsung Galaxy Note 4	1024	68.2	73.5	84.4
Motorola Moto X	128	53.7	53.9	54.8
Motorola Moto X	256	55.1	55.3	55.9
Motorola Moto X	512	56.0	56.1	56.5
Motorola Moto X	1024	56.5	56.6	56.9

Table 7.13: AES-CTR OpenSSL Decryption Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	6.6	6.8	6.9
HTC One X	256	6.8	6.9	7.0
HTC One X	512	6.7	6.9	7.0
HTC One X	1024	6.8	7.0	7.1
Samsung Galaxy S4	128	13.1	13.8	17.0
Samsung Galaxy S4	256	12.6	13.4	15.9
Samsung Galaxy S4	512	13.1	13.7	15.3
Samsung Galaxy S4	1024	13.0	14.1	17.5
Samsung Galaxy Note 4	128	15.7	16.1	21.8
Samsung Galaxy Note 4	256	15.6	15.9	18.9
Samsung Galaxy Note 4	512	14.2	16.0	18.5
Samsung Galaxy Note 4	1024	15.1	16.0	22.4
Motorola Moto X	128	8.3	8.5	8.6
Motorola Moto X	256	16.8	21.8	2.4
Motorola Moto X	512	8.6	8.6	8.6
Motorola Moto X	1024	8.6	8.6	8.7

Table 7.14: AES-CTR BoringSSL Encryption Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	6.6	6.8	6.9
HTC One X	256	6.7	6.9	7.0
HTC One X	512	6.8	6.9	7.0
HTC One X	1024	6.8	6.9	7.0
Samsung Galaxy S4	128	12.3	12.8	13.0
Samsung Galaxy S4	256	12.6	13.1	14.0
Samsung Galaxy S4	512	12.8	13.2	14.3
Samsung Galaxy S4	1024	13.1	13.5	14.3
Samsung Galaxy Note 4	128	15.6	15.9	16.1
Samsung Galaxy Note 4	256	14.9	15.4	15.7
Samsung Galaxy Note 4	512	12.8	16.3	16.9
Samsung Galaxy Note 4	1024	13.1	14.8	15.4
Motorola Moto X	128	8.3	8.4	8.5
Motorola Moto X	256	16.4	21.1	22.1
Motorola Moto X	512	8.5	8.6	8.6
Motorola Moto X	1024	8.5	8.6	8.6

Table 7.15: AES-CTR BoringSSL Decryption Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	4.7	4.8	4.9
HTC One X	256	4.7	4.8	4.9
HTC One X	512	4.7	4.8	5.0
HTC One X	1024	4.8	4.9	4.9
Samsung Galaxy S4	128	10.4	11.3	12.0
Samsung Galaxy S4	256	9.7	10.9	11.9
Samsung Galaxy S4	512	9.7	10.6	11.9
Samsung Galaxy S4	1024	10.3	11.0	11.8
Samsung Galaxy Note 4	128	12.2	13.5	17.1
Samsung Galaxy Note 4	256	12.1	12.4	12.9
Samsung Galaxy Note 4	512	11.0	12.4	13.1
Samsung Galaxy Note 4	1024	11.6	13.4	16.6
Motorola Moto X	128	7.2	7.2	7.3
Motorola Moto X	256	16.3	17.3	18.6
Motorola Moto X	512	7.2	72.7	7.3
Motorola Moto X	1024	16.4	17.7	18.9

Table 7.16: AES-CTR wolfCrypt Encryption Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	4.7	4.8	4.9
HTC One X	256	4.7	4.8	4.9
HTC One X	512	4.7	4.8	4.9
HTC One X	1024	4.8	4.9	4.9
Samsung Galaxy S4	128	9.9	10.6	11.2
Samsung Galaxy S4	256	9.8	10.4	11.0
Samsung Galaxy S4	512	9.5	10.2	11.1
Samsung Galaxy S4	1024	9.9	10.5	11.6
Samsung Galaxy Note 4	128	11.8	12.3	17.1
Samsung Galaxy Note 4	256	12.2	12.4	12.6
Samsung Galaxy Note 4	512	12.2	12.3	12.5
Samsung Galaxy Note 4	1024	12.0	12.4	12.6
Motorola Moto X	128	7.2	7.2	7.2
Motorola Moto X	256	16.3	16.4	16.6
Motorola Moto X	512	7.2	7.2	7.2
Motorola Moto X	1024	16.2	16.3	16.4

Table 7.17: AES-CTR wolfCrypt Decryption Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	12.5	12.9	13.1
HTC One X	256	25.0	25.7	26.1
HTC One X	512	49.9	51.6	53.6
HTC One X	1024	101.0	103.1	104.4
Samsung Galaxy S4	128	28.2	31.1	33.7
Samsung Galaxy S4	256	55.8	60.6	65.0
Samsung Galaxy S4	512	117.4	127.6	136.9
Samsung Galaxy S4	1024	236.8	261.2	280.0
Samsung Galaxy Note 4	128	33.1	35.5	48.5
Samsung Galaxy Note 4	256	65.5	73.8	97.3
Samsung Galaxy Note 4	512	139.2	159.3	193.6
Samsung Galaxy Note 4	1024	277.1	297.0	385.5
Motorola Moto X	128	50.5	66.1	75.2
Motorola Moto X	256	42.5	42.9	43.0
Motorola Moto X	512	85.6	85.7	86.7
Motorola Moto X	1024	400.4	529.3	605.6

Table 7.18: AES-CTR mbedTLS Encryption Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	12.5	13.0	13.2
HTC One X	256	25.4	25.9	26.2
HTC One X	512	50.8	52.1	52.6
HTC One X	1024	100.9	103.5	106.8
Samsung Galaxy S4	128	28.6	30.7	32.5
Samsung Galaxy S4	256	58.7	61.1	64.3
Samsung Galaxy S4	512	115.9	123.2	131.3
Samsung Galaxy S4	1024	231.7	248.4	272.0
Samsung Galaxy Note 4	128	31.0	36.6	48.9
Samsung Galaxy Note 4	256	68.5	74.6	96.8
Samsung Galaxy Note 4	512	135.5	147.6	192.2
Samsung Galaxy Note 4	1024	266.7	295.6	385.1
Motorola Moto X	128	66.2	66.3	66.4
Motorola Moto X	256	42.8	42.9	42.9
Motorola Moto X	512	85.5	85.6	85.6
Motorola Moto X	1024	170.2	171.6	183.2

Table 7.19: AES-CTR mbedTLS Decryption Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	0.06	0.07	0.07
HTC One X	256	0.07	0.07	0.07
HTC One X	512	0.07	0.07	0.07
HTC One X	1024	0.07	0.07	0.08
Samsung Galaxy S4	128	0.5	0.6	0.6
Samsung Galaxy S4	256	0.6	0.6	0.7
Samsung Galaxy S4	512	0.7	0.7	0.7
Samsung Galaxy S4	1024	0.7	0.7	0.8
Samsung Galaxy Note 4	128	0.008	0.09	0.09
Samsung Galaxy Note 4	256	0.08	0.09	0.09
Samsung Galaxy Note 4	512	0.09	0.1	0.1
Samsung Galaxy Note 4	1024	0.09	0.09	0.09
Motorola Moto X	128	2.1	2.2	2.2
Motorola Moto X	256	2.5	3.3	6.5
Motorola Moto X	512	2.7	2.8	2.8
Motorola Moto X	1024	2.9	2.9	2.9

Table 7.20: AES-CTR Bouncy Castle Encryption Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	0.08	0.08	0.08
HTC One X	256	0.08	0.09	0.09
HTC One X	512	0.08	0.09	0.09
HTC One X	1024	0.08	0.09	0.09
Samsung Galaxy S4	128	1.1	1.1	1.3
Samsung Galaxy S4	256	1.2	1.3	1.4
Samsung Galaxy S4	512	1.4	1.4	1.5
Samsung Galaxy S4	1024	1.4	1.5	1.6
Samsung Galaxy Note 4	128	0.09	0.1	0.1
Samsung Galaxy Note 4	256	0.1	0.1	0.1
Samsung Galaxy Note 4	512	0.1	0.1	0.1
Samsung Galaxy Note 4	1024	0.1	0.1	0.1
Motorola Moto X	128	4.0	4.1	4.5
Motorola Moto X	256	5.1	5.2	5.5
Motorola Moto X	512	5.7	5.7	5.9
Motorola Moto X	1024	6.2	6.2	6.2

Table 7.21: AES-CTR Bouncy Castle Decryption Results

7.4 AES-GCM Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	7.8	8.0	8.1
HTC One X	256	7.9	8.1	8.2
HTC One X	512	8.0	8.2	8.3
HTC One X	1024	8.1	8.2	8.4
Samsung Galaxy S4	128	17.0	18.2	19.8
Samsung Galaxy S4	256	17.0	17.6	18.1
Samsung Galaxy S4	512	17.4	19.0	20.8
Samsung Galaxy S4	1024	16.6	17.7	19.4
Samsung Galaxy Note 4	128	19.2	22.8	27.0
Samsung Galaxy Note 4	256	18.9	20.5	26.4
Samsung Galaxy Note 4	512	18.4	20.7	26.6
Samsung Galaxy Note 4	1024	19.4	20.3	27.6
Motorola Moto X	128	10.6	10.6	10.7
Motorola Moto X	256	10.8	10.8	10.8
Motorola Moto X	512	10.8	10.8	10.9
Motorola Moto X	1024	10.9	10.9	10.9

Table 7.22: AES-GCM BoringSSL Encryption Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	5.4	5.5	5.6
HTC One X	256	6.3	6.5	6.6
HTC One X	512	7.0	7.1	7.2
HTC One X	1024	7.3	7.4	7.6
Samsung Galaxy S4	128	11.6	12.7	13.3
Samsung Galaxy S4	256	13.7	14.7	15.5
Samsung Galaxy S4	512	14.7	15.7	16.8
Samsung Galaxy S4	1024	16.0	16.7	17.4
Samsung Galaxy Note 4	128	13.5	14.5	18.7
Samsung Galaxy Note 4	256	15.5	16.6	21.3
Samsung Galaxy Note 4	512	16.9	17.4	17.6
Samsung Galaxy Note 4	1024	18.1	18.5	18.8
Motorola Moto X	128	8.2	8.2	8.2
Motorola Moto X	256	9.4	9.4	9.4
Motorola Moto X	512	10.0	10.1	10.1
Motorola Moto X	1024	10.5	10.5	10.5

Table 7.23: AES-GCM BoringSSL Decryption Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	0.1	0.1	0.1
HTC One X	256	0.1	0.1	0.1
HTC One X	512	0.1	0.1	0.1
HTC One X	1024	0.1	0.1	0.1
Samsung Galaxy S4	128	1.4	1.5	1.6
Samsung Galaxy S4	256	1.5	1.6	1.7
Samsung Galaxy S4	512	1.5	1.6	1.7
Samsung Galaxy S4	1024	1.6	1.6	1.7
Samsung Galaxy Note 4	128	0.1	0.1	0.1
Samsung Galaxy Note 4	256	0.1	0.1	0.1
Samsung Galaxy Note 4	512	0.1	0.1	0.1
Samsung Galaxy Note 4	1024	0.1	0.1	0.1
Motorola Moto X	128	4.5	4.6	4.7
Motorola Moto X	256	5.0	5.1	5.1
Motorola Moto X	512	5.1	5.2	5.2
Motorola Moto X	1024	5.2	5.3	5.3

Table 7.24: AES-GCM BouncyCastle Encryption Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	0.1	0.1	0.1
HTC One X	256	0.1	0.1	0.1
HTC One X	512	0.1	0.1	0.1
HTC One X	1024	0.1	0.1	0.1
Samsung Galaxy S4	128	1.5	1.6	1.7
Samsung Galaxy S4	256	1.6	0.1	0.1
Samsung Galaxy S4	512	1.6	0.1	0.1
Samsung Galaxy S4	1024	1.6	0.1	0.1
Samsung Galaxy Note 4	128	0.1	0.1	0.1
Samsung Galaxy Note 4	256	0.1	0.1	0.1
Samsung Galaxy Note 4	512	0.1	0.1	0.1
Samsung Galaxy Note 4	1024	0.1	0.1	0.1
Motorola Moto X	128	4.6	4.6	4.6
Motorola Moto X	256	5.0	5.0	5.0
Motorola Moto X	512	5.1	5.1	5.2
Motorola Moto X	1024	5.3	5.3	5.3

Table 7.25: AES-GCM Bouncy Castle Decryption Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	3.7	3.8	3.9
HTC One X	256	7.4	7.7	7.8
HTC One X	512	15.0	15.4	15.6
HTC One X	1024	30.2	30.9	31.3
Samsung Galaxy S4	128	8.7	9.3	9.9
Samsung Galaxy S4	256	17.5	18.4	19.7
Samsung Galaxy S4	512	34.8	37.0	40.1
Samsung Galaxy S4	1024	69.9	74.5	78.4
Samsung Galaxy Note 4	128	9.7	12.5	13.6
Samsung Galaxy Note 4	256	18.9	24.0	26.5
Samsung Galaxy Note 4	512	38.2	50.5	54.4
Samsung Galaxy Note 4	1024	76.4	90.0	110.1
Motorola Moto X	128	5.9	59.3	5.9
Motorola Moto X	256	11.8	11.8	12.0
Motorola Moto X	512	23.7	23.7	23.8
Motorola Moto X	1024	47.3	47.4	48.0

Table 7.26: AES-GCM mbedTLS Encryption Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	3.7	3.8	3.8
HTC One X	256	7.5	7.7	7.7
HTC One X	512	15.0	15.3	15.5
HTC One X	1024	30.1	30.7	31.1
Samsung Galaxy S4	128	8.3	89.7	9.7
Samsung Galaxy S4	256	17.2	18.2	19.4
Samsung Galaxy S4	512	34.0	36.5	39.5
Samsung Galaxy S4	1024	67.0	71.6	76.9
Samsung Galaxy Note 4	128	9.5	10.9	13.3
Samsung Galaxy Note 4	256	18.9	22.1	26.8
Samsung Galaxy Note 4	512	37.5	44.9	54.0
Samsung Galaxy Note 4	1024	88.7	75.6	106.7
Motorola Moto X	128	5.8	5.8	5.8
Motorola Moto X	256	11.7	11.7	11.7
Motorola Moto X	512	23.1	23.4	23.4
Motorola Moto X	1024	46.4	46.8	46.8

Table 7.27: AES-GCM mbedTLS Decryption Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	14.9	15.3	15.6
HTC One X	256	15.2	15.5	15.6
HTC One X	512	15.2	15.6	15.8
HTC One X	1024	15.3	15.7	17.9
Samsung Galaxy S4	128	35.4	38.5	42.3
Samsung Galaxy S4	256	34.9	38.2	41.3
Samsung Galaxy S4	512	35.7	39.0	43.1
Samsung Galaxy S4	1024	36.7	39.7	45.1
Samsung Galaxy Note 4	128	40.1	42.0	56.5
Samsung Galaxy Note 4	256	25.0	41.4	56.8
Samsung Galaxy Note 4	512	41.0	44.4	59.6
Samsung Galaxy Note 4	1024	41.0	43.4	57.7
Motorola Moto X	128	29.7	29.8	32.4
Motorola Moto X	256	30.3	42.8	45.1
Motorola Moto X	512	30.6	31.4	45.0
Motorola Moto X	1024	30.4	34.3	45.5

Table 7.28: AES-GCM OpenSSL Encryption Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	10.9	11.2	11.3
HTC One X	256	12.9	13.2	13.3
HTC One X	512	13.4	14.3	14.5
HTC One X	1024	14.7	15.1	15.3
Samsung Galaxy S4	128	24.3	25.7	27.0
Samsung Galaxy S4	256	26.1	28.3	30.1
Samsung Galaxy S4	512	31.5	32.9	34.5
Samsung Galaxy S4	1024	30.8	33.1	34.8
Samsung Galaxy Note 4	128	15.8	23.4	35.4
Samsung Galaxy Note 4	256	20.3	26.6	40.3
Samsung Galaxy Note 4	512	22.5	32.6	49.7
Samsung Galaxy Note 4	1024	23.0	34.2	49.7
Motorola Moto X	128	21.5	21.6	21.6
Motorola Moto X	256	25.4	25.5	25.5
Motorola Moto X	512	27.9	27.9	28.0
Motorola Moto X	1024	29.3	29.4	29.4

Table 7.29: AES-GCM OpenSSL Decryption Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	0.5	0.5	0.5
HTC One X	256	1.1	1.2	1.2
HTC One X	512	2.0	2.1	2.2
HTC One X	1024	4.0	4.2	4.4
Samsung Galaxy S4	128	1.4	1.5	1.8
Samsung Galaxy S4	256	2.5	2.8	3.3
Samsung Galaxy S4	512	5.6	6.2	7.6
Samsung Galaxy S4	1024	11.4	12.4	15.3
Samsung Galaxy Note 4	128	1.8	2.2	2.4
Samsung Galaxy Note 4	256	3.6	4.1	5.0
Samsung Galaxy Note 4	512	7.1	9.2	9.9
Samsung Galaxy Note 4	1024	14.4	18.4	19.9
Motorola Moto X	128	1.0	1.1	1.1
Motorola Moto X	256	2.2	2.2	2.2
Motorola Moto X	512	4.4	4.4	4.4
Motorola Moto X	1024	8.7	8.8	8.9

Table 7.30: AES-GCM wolfCrypt Encryption Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	0.5	0.5	0.5
HTC One X	256	1.1	1.2	1.2
HTC One X	512	2.0	2.1	2.2
HTC One X	1024	4.0	4.2	4.5
Samsung Galaxy S4	128	1.2	1.4	1.5
Samsung Galaxy S4	256	2.4	2.7	3.0
Samsung Galaxy S4	512	5.5	5.8	6.0
Samsung Galaxy S4	1024	11.2	11.7	1.2
Samsung Galaxy Note 4	128	1.7	2.1	2.4
Samsung Galaxy Note 4	256	3.5	4.0	4.8
Samsung Galaxy Note 4	512	7.2	8.4	9.8
Samsung Galaxy Note 4	1024	14.0	16.8	19.4
Motorola Moto X	128	1.0	1.1	1.1
Motorola Moto X	256	2.1	2.1	2.2
Motorola Moto X	512	4.3	4.3	4.3
Motorola Moto X	1024	8.7	8.7	9.0

Table 7.31: AES-GCM wolfCrypt Decryption Results

7.5 AES-OFB Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	0.3	0.3	0.3
HTC One X	256	0.4	0.4	0.4
HTC One X	512	0.4	0.4	0.4
HTC One X	1024	0.4	0.4	0.4
Samsung Galaxy S4	128	0.8	0.9	0.9
Samsung Galaxy S4	256	1.0	1.1	1.1
Samsung Galaxy S4	512	1.1	1.2	1.2
Samsung Galaxy S4	1024	1.2	1.2	1.3
Samsung Galaxy Note 4	128	0.5	0.6	0.6
Samsung Galaxy Note 4	256	0.5	0.6	0.7
Samsung Galaxy Note 4	512	0.6	0.6	0.7
Samsung Galaxy Note 4	1024	0.6	0.7	0.7
Motorola Moto X	128	2.3	2.8	5.3
Motorola Moto X	256	2.6	2.6	2.6
Motorola Moto X	512	2.8	2.8	2.8
Motorola Moto X	1024	2.9	2.9	2.9

Table 7.32: AES-OFB Bouncy Castle Encryption Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
Mobile Phone	Size of string	Worst Case (Bytes/second)	Average (Bytes/second)	Best Case (Bytes/second)
HTC One X	128	2.5	2.6	2.7
HTC One X	256	3.0	3.1	3.1
HTC One X	512	3.3	3.4	3.5
HTC One X	1024	3.4	3.5	3.6
Samsung Galaxy S4	128	2.6	2.6	2.7
Samsung Galaxy S4	256	3.2	3.5	3.6
Samsung Galaxy S4	512	3.7	3.9	4.1
Samsung Galaxy S4	1024	3.3	4.2	4.6
Samsung Galaxy Note 4	128	3.2	3.7	4.0
Samsung Galaxy Note 4	256	4.2	4.5	5.4
Samsung Galaxy Note 4	512	5.1	5.2	5.2
Samsung Galaxy Note 4	1024	4.9	5.4	5.9
Motorola Moto X	128	4.5	4.5	4.6
Motorola Moto X	256	5.3	5.4	5.4
Motorola Moto X	512	5.7	5.7	5.7
Motorola Moto X	1024	6.0	6.0	6.0

Table 7.33: AES-OFB Bouncy Castle Decryption Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	3.0	3.1	3.1
HTC One X	256	3.0	3.1	3.1
HTC One X	512	3.0	3.1	3.2
HTC One X	1024	3.1	3.1	3.2
Samsung Galaxy S4	128	6.4	7.2	7.8
Samsung Galaxy S4	256	6.7	7.3	8.3
Samsung Galaxy S4	512	7.1	7.9	8.9
Samsung Galaxy S4	1024	7.6	8.2	9.0
Samsung Galaxy Note 4	128	8.6	9.5	12.1
Samsung Galaxy Note 4	256	8.8	9.4	12.5
Samsung Galaxy Note 4	512	8.8	9.8	12.8
Samsung Galaxy Note 4	1024	8.9	9.7	12.5
Motorola Moto X	128	5.9	5.9	5.9
Motorola Moto X	256	6.0	6.0	6.1
Motorola Moto X	512	6.1	6.1	6.1
Motorola Moto X	1024	6.1	6.1	6.2

Table 7.34: AES-OFB OpenSSL Encryption Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	2419.1	2468.7	2499.1
HTC One X	256	4845.7	4945.9	4999.1
HTC One X	512	9676.5	9876.6	9998.0
HTC One X	1024	19025.9	19809.7	20001.7
Samsung Galaxy S4	128	2330.6	2700.8	3035.0
Samsung Galaxy S4	256	4317.3	5289.7	5925.1
Samsung Galaxy S4	512	8837.1	10662.7	12021.1
Samsung Galaxy S4	1024	17194.9	20257.1	23880.4
Samsung Galaxy Note 4	128	1244.1	3203.0	3427.3
Samsung Galaxy Note 4	256	4421.2	4616.0	4947.9
Samsung Galaxy Note 4	512	8918.9	9257.9	9695.3
Samsung Galaxy Note 4	1024	81.0	14470.7	19497.0
Motorola Moto X	128	3650.2	3666.4	3668.6
Motorola Moto X	256	7325.7	7334.7	7337.4
Motorola Moto X	512	14519.6	14665.6	14676.6
Motorola Moto X	1024	29320.2	29339.8	29348.6

Table 7.35: AES-OFB OpenSSL Decryption Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	8.5	8.7	8.7
HTC One X	256	8.6	8.7	8.9
HTC One X	512	8.6	8.8	9.0
HTC One X	1024	8.7	8.8	9.0
Samsung Galaxy S4	128	26.7	28.4	30.9
Samsung Galaxy S4	256	26.1	27.8	29.4
Samsung Galaxy S4	512	26.1	27.8	29.6
Samsung Galaxy S4	1024	26.1	27.7	29.8
Samsung Galaxy Note 4	128	25.5	31.8	36.2
Samsung Galaxy Note 4	256	25.5	30.8	35.5
Samsung Galaxy Note 4	512	25.3	29.5	35.9
Samsung Galaxy Note 4	1024	26.1	30.9	37.1
Motorola Moto X	128	12.7	12.7	12.8
Motorola Moto X	256	12.9	12.9	13.0
Motorola Moto X	512	13.0	13.0	13.1
Motorola Moto X	1024	12.9	12.9	13.0

Table 7.36: AES-OFB BoringSSL Encryption Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	8.0	8.2	8.3
HTC One X	256	8.0	8.2	8.4
HTC One X	512	8.1	8.3	8.4
HTC One X	1024	8.2	8.4	8.4
Samsung Galaxy S4	128	4290.1	4436.3	4573.1
Samsung Galaxy S4	256	8642.7	9007.5	9144.3
Samsung Galaxy S4	512	17069.0	17850.3	18309.2
Samsung Galaxy S4	1024	32107.8	34255.3	36179.6
Samsung Galaxy Note 4	128	1768.9	3052.5	3854.9
Samsung Galaxy Note 4	256	3495.6	5822.0	7773.4
Samsung Galaxy Note 4	512	7147.9	12465.1	15401.2
Samsung Galaxy Note 4	1024	16738.8	26759.4	32508.0
Motorola Moto X	128	2410.6	2413.2	2414.0
Motorola Moto X	256	4823.0	4834.4	5083.1
Motorola Moto X	512	9634.8	9664.3	10017.7
Motorola Moto X	1024	19201.2	19342.1	20669.5

Table 7.37: AES-OFB BoringSSL Decryption Results

7.6 MD-5 Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	24.8	25.6	26.0
HTC One X	256	25.5	26.3	26.8
HTC One X	512	26.1	26.8	27.2
HTC One X	1024	26.5	27.0	28.2
Samsung Galaxy S4	128	47.5	50.2	53.0
Samsung Galaxy S4	256	47.0	49.8	53.4
Samsung Galaxy S4	512	47.7	51.0	53.8
Samsung Galaxy S4	1024	48.9	51.3	53.9
Samsung Galaxy Note 4	128	51.0	57.6	73.2
Samsung Galaxy Note 4	256	51.9	60.9	73.8
Samsung Galaxy Note 4	512	52.1	61.3	74.6
Samsung Galaxy Note 4	1024	52.9	66.3	75.4
Motorola Moto X	128	44.5	45.1	45.2
Motorola Moto X	256	45.3	46.0	46.3
Motorola Moto X	512	45.7	46.4	46.8
Motorola Moto X	1024	45.9	46.6	47.6

Table 7.38: MD-5 BoringSSL Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	0.3	5.	11.4
HTC One X	256	0.3	6.1	11.9
HTC One X	512	0.4	6.3	12.2
HTC One X	1024	0.4	6.3	12.2
Samsung Galaxy S4	128	2.7	52.6	102.0
Samsung Galaxy S4	256	3.8	56.0	106.5
Samsung Galaxy S4	512	3.9	57.1	108.5
Samsung Galaxy S4	1024	4.1	62.8	121.4
Samsung Galaxy Note 4	128	0.6	9.4	18.1
Samsung Galaxy Note 4	256	0.6	10.3	19.7
Samsung Galaxy Note 4	512	0.6	9.4	17.8
Samsung Galaxy Note 4	1024	0.7	9.4	17.8
Motorola Moto X	128	10.3	153.6	296.7
Motorola Moto X	256	14.7	218.6	421.8
Motorola Moto X	512	17.4	264.4	511.4
Motorola Moto X	1024	20.6	313.6	605.7

Table 7.39: MD-5 Bouncy Castle Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	21.8	22.4	22.7
HTC One X	256	28.3	29.3	29.7
HTC One X	512	33.7	34.4	35.0
HTC One X	1024	37.1	37.9	38.4
Samsung Galaxy S4	128	46.6	50.3	53.8
Samsung Galaxy S4	256	57.6	62.6	66.6
Samsung Galaxy S4	512	66.0	72.7	77.3
Samsung Galaxy S4	1024	74.4	79.1	86.7
Samsung Galaxy Note 4	128	50.1	66.8	70.1
Samsung Galaxy Note 4	256	64.6	80.0	906.8
Samsung Galaxy Note 4	512	76.0	94.1	107.4
Samsung Galaxy Note 4	1024	83.6	99.2	117.7
Motorola Moto X	128	37.2	37.2	37.4
Motorola Moto X	256	48.7	48.7	48.8
Motorola Moto X	512	57.8	57.9	58.0
Motorola Moto X	1024	63.8	63.9	64.0

Table 7.40: MD-5 mbedTLS Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	102.4	104.5	105.9
HTC One X	256	105.1	107.8	109.3
HTC One X	512	106.5	110.1	111.9
HTC One X	1024	109.1	111.3	112.8
Samsung Galaxy S4	128	176.6	190.6	211.2
Samsung Galaxy S4	256	196.0	207.3	217.5
Samsung Galaxy S4	512	194.8	202.8	212.6
Samsung Galaxy S4	1024	193.7	202.1	211.8
Samsung Galaxy Note 4	128	206.6	226.6	294.3
Samsung Galaxy Note 4	256	213.9	257.0	303.9
Samsung Galaxy Note 4	512	216.1	232.6	307.1
Samsung Galaxy Note 4	1024	218.5	228.3	308.2
Motorola Moto X	128	204.6	244.7	321.1
Motorola Moto X	256	215.0	215.6	222.0
Motorola Moto X	512	219.1	221.1	230.6
Motorola Moto X	1024	223.1	223.6	226.1

Table 7.41: MD-5 OpenSSL Results

Mobile Phone	Size of string (Bytes)	Worst Case (MBytes/second)	Average (MBytes/second)	Best Case (MBytes/second)
HTC One X	128	13.6	13.9	14.1
HTC One X	256	18.1	18.4	18.7
HTC One X	512	21.1	22.1	22.4
HTC One X	1024	23.3	24.4	24.9
Samsung Galaxy S4	128	21.6	22.6	23.9
Samsung Galaxy S4	256	31.5	34.1	36.0
Samsung Galaxy S4	512	37.7	41.1	45.0
Samsung Galaxy S4	1024	45.0	48.8	53.1
Samsung Galaxy Note 4	128	21.3	23.4	27.5
Samsung Galaxy Note 4	256	32.1	38.0	42.7
Samsung Galaxy Note 4	512	42.6	44.5	58.5
Samsung Galaxy Note 4	1024	50.6	53.3	70.6
Motorola Moto X	128	21.3	21.4	21.7
Motorola Moto X	256	28.2	28.2	28.2
Motorola Moto X	512	33.6	33.7	33.9
Motorola Moto X	1024	37.2	37.4	42.1

Table 7.42: MD-5 wolfCrypt Results

7.7 RSA Results

Mobile Phone	Library	Worst Case (kBytes/second)	Average (kBytes/second)	Best Case (kBytes/second)
HTC One X	Bouncy Castle	20.8	21.4	22.5
HTC One X	BoringSSL	71.9	73.5	75.2
HTC One X	mbedTLS	10.4	10.6	10.8
HTC One X	OpenSSL	106.9	10.9	110.7
HTC One X	wolfCrypt	4.6	4.7	4.8
Samsung Galaxy S4	Bouncy Castle	100.4	11.2	120.1
Samsung Galaxy S4	BoringSSL	141.9	15.8	181.8
Samsung Galaxy S4	mbedTLS	28.1	29.2	32.8
Samsung Galaxy S4	OpenSSL	157.1	17.4	203.3
Samsung Galaxy S4	wolfCrypt	12.3	13.5	16.0
Samsung Galaxy Note 4	Bouncy Castle	25.9	30.1	32.8
Samsung Galaxy Note 4	BoringSSL	186.7	193.7	260.8
Samsung Galaxy Note 4	mbedTLS	34.0	37.1	38.5
Samsung Galaxy Note 4	OpenSSL	203.3	212.5	286.6
Samsung Galaxy Note 4	wolfCrypt	15.6	16.1	21.9
Motorola Moto X	Bouncy Castle	149.0	157.1	163.0
Motorola Moto X	BoringSSL	153.4	164.1	171.6
Motorola Moto X	mbedTLS	17.3	17.3	17.6
Motorola Moto X	OpenSSL	219.1	219.5	221.0
Motorola Moto X	wolfCrypt	8.1	8.1	8.1

Table 7.43: RSA 128 bytes Encryption Results

Mobile Phone	Library	Worst Case (kBytes/second)	Average (kBytes/second)	Best Case (kBytes/second)
HTC One X	Bouncy Castle	2.4	2.5	2.5
HTC One X	BoringSSL	3.3	3.3	3.4
HTC One X	mbedTLS	0.2	0.2	0.2
HTC One X	OpenSSL	5.9	6.0	6.1
HTC One X	wolfCrypt	0.2	0.2	0.2
Samsung Galaxy S4	Bouncy Castle	4.4	4.6	4.9
Samsung Galaxy S4	BoringSSL	6.5	7.0	7.5
Samsung Galaxy S4	mbedTLS	0.7	0.7	0.8
Samsung Galaxy S4	OpenSSL	8.7	9.3	10.2
Samsung Galaxy S4	wolfCrypt	0.7	0.7	0.8
Samsung Galaxy Note 4	Bouncy Castle	3.1	3.2	4.2
Samsung Galaxy Note 4	BoringSSL	8.1	8.3	8.5
Samsung Galaxy Note 4	mbedTLS	0.9	0.9	0.9
Samsung Galaxy Note 4	OpenSSL	11.4	11.6	11.8
Samsung Galaxy Note 4	wolfCrypt	0.9	0.9	0.9
Motorola Moto X	Bouncy Castle	6.8	7.0	7.1
Motorola Moto X	BoringSSL	7.0	7.1	7.2
Motorola Moto X	mbedTLS	0.4	0.4	0.4
Motorola Moto X	OpenSSL	12.1	12.1	12.2
Motorola Moto X	wolfCrypt	0.4	0.5	0.5

Table 7.44: RSA 128 bytes Decryption Results

7.8 Key Agreements algorithms

Mobile Phone	Library	Worst Case (Key Agreements/second)	Average (Key Agreements/second)	Best Case (Key Agreements/second)
HTC One X	Bouncy Castle	105013	109172	113277
HTC One X	BoringSSL	4	4	4
HTC One X	mbedTLS	0.74	0.77	0.78
HTC One X	OpenSSL	8965671	9239238	9364808
HTC One X	wolfCrypt	4	4	5
Samsung Galaxy S4	Bouncy Castle	84127	86754	89437
Samsung Galaxy S4	BoringSSL	9	10	10
Samsung Galaxy S4	mbedTLS	2	2	2
Samsung Galaxy S4	OpenSSL	14415366	17232152	18801422
Samsung Galaxy S4	wolfCrypt	13	15	16
Samsung Galaxy Note 4	Bouncy Castle	149942	172289	190114
Samsung Galaxy Note 4	BoringSSL	10	14	15
Samsung Galaxy Note 4	mbedTLS	2	3	3
Samsung Galaxy Note 4	OpenSSL	18249320	23173245	28673990
Samsung Galaxy Note 4	wolfCrypt	16	18	22
Motorola Moto X	Bouncy Castle	118015	119875	122024
Motorola Moto X	BoringSSL	9	9	9
Motorola Moto X	mbedTLS	1	1	1
Motorola Moto X	OpenSSL	8477281	8625480	8646604
Motorola Moto X	wolfCrypt	8	8	8

Table 7.45: Diffie Hellman Results

Mobile Phone	Library	Worst Case (Key Agreements/second)	Average (Key Agreements/second)	Best Case (Key Agreements/second)
HTC One X	Bouncy Castle	67196	70593	73714
HTC One X	BoringSSL	20	20	21
HTC One X	mbedTLS	13	13	13
HTC One X	OpenSSL	198	202	206
HTC One X	wolfCrypt	16	16	16
Samsung Galaxy S4	Bouncy Castle	167479	176717	189147
Samsung Galaxy S4	BoringSSL	38	43	49
Samsung Galaxy S4	mbedTLS	30	32	35
Samsung Galaxy S4	OpenSSL	330	348	403
Samsung Galaxy S4	wolfCrypt	43	47	52
Samsung Galaxy Note 4	Bouncy Castle	81322	93339	97875
Samsung Galaxy Note 4	BoringSSL	49	53	66
Samsung Galaxy Note 4	mbedTLS	36	38	51
Samsung Galaxy Note 4	OpenSSL	401	413	481
Samsung Galaxy Note 4	wolfCrypt	49	52	68
Motorola Moto X	Bouncy Castle	314708	316869	318549
Motorola Moto X	BoringSSL	35	37	39
Motorola Moto X	mbedTLS	25	25	25
Motorola Moto X	OpenSSL	339	339	340
Motorola Moto X	wolfCrypt	31	31	31

Table 7.46: Elliptic Curve Diffie Hellman Results

Bibliography

[04] "Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP)". In: (2004). URL: https://tools.ietf.org/html/rfc3686.

[17] "The top 10 Android Statistics marketers need to know". In: *Mediakix* (2017).

[auta] Unknown author. "Algorithms explained: Diffie-Hellman". In: (). URL: https://hackernoon.com/algorithms-explained-diffie-hellman-1034210d5100.

[autb] Unknown author. *Android Studio*. URL: https://developer.android.com/studio/.

[autc] Unknown author. *BoringSSL*. URL: https://opensource.google.com/projects/boringssl.

[autd] Unknown author. *BoringSSL GitHub Repository*. URL: https://github.com/google/boringssl/blob/master/BUILDING.md.

[aute] Unknown author. *Bouncy Castle*. URL: https://www.bouncycastle.org/%22.

[autf] Unknown author. *Cmake*. URL: https://cmake.org/.

[autg] Unknown author. *mbedTLS*. URL: https://tls.mbed.org/.

[auth] Unknown author. *mbedTLS GitHub Repository*. URL: https://github.com/ARMmbed/mbedtls.

[auti] Unknown author. *Motorola Moto X*. URL: https://www.gsmarena.com/motorola_moto_x_style-7229.php.

[autj] Unknown author. *OpenSSL*. URL: https://www.openssl.org/.

[autk] Unknown author. *OpenSSL Android tutorial*. URL: https://wiki.openssl.org/index.php/Android.

[autl] Unknown author. *Optimize for devices running Android (Go edition)*. URL: https://developer.android.com/docs/quality-guidelines/building-for-billions-device-capacity#androidgo.

[autm] Unknown author. *Samsung Galaxy Note 4*. URL: https://gadgets.ndtv.com/samsung-galaxy-note-4-1937.

[autn] Unknown author. *Samsung Galaxy S4*. URL: https://www.androidcentral.com/samsung-galaxy-s4-specs.

[auto] Unknown author. *Spongy Castle*. URL: https://rtyley.github.io/spongycastle/.

[autp] Unknown author. *What is the difference between CBC and GCM mode*. URL: https://crypto.stackexchange.com/questions/2310/what-is-the-difference-between-cbc-and-gcm-mode.

[autq] Unknown author. *wolfCrypt*. URL: `https://www.wolfssl.com/products/wolfcrypt/`.

[autr] Unknown author. *wolfCrypt GitHub Repository*. URL: `https://github.com/wolfSSL/wolfcrypt-jni`.

[aut08] Unknown author. "Additional Diffie-Hellman Groups for Use with IETF Standards". In: (2008). URL: `https://tools.ietf.org/html/rfc5114`.

[aut17] Unknown author. *Mobile(Android) sardware stats*. 2017. URL: `https://web.archive.org/web/20170808222202/http://hwstats.unity3d.com:80/mobile/cpu-android.html`.

[Bel] Steven M. Bellovin. *Modes of operation*. URL: `https://crypto.stackexchange.com/questions/2310/what-is-the-difference-between-cbc-and-gcm-mode`.

[Blo18] Bloomberg. "Facebook Cambridge Analytica Scandal: 10 Questions Answered". In: *Fortune* (2018). URL: `http://fortune.com/2018/04/10/facebook-cambridge-analytica-what-happened/`.

[Cor18] Caroline Corrigan. *Why your phone gets hot and how to fix it*. 2018. URL: `https://www.avg.com/en/signal/why-your-phone-gets-hot-and-how-to-fix-it`.

[Din+15] Daniel Dinu et al. "FELICS – Fair Evaluation of Lightweight Cryptographic Systems". In: *Univerisy of Luxembourg* (2015).

[FW86] Philip J. Fleming and John J. Wallace. "How not to lie with statistics: the correct way to summarize benchmark results". In: *Communications of the ACM* (1986). URL: `http://portal.acm.org/citation.cfm?doid=5666.5673`.

[Gon+15] David González et al. "Evaluation of Cryptographic Capabilities for the Android Platform". In: *Network Engineering Department, Universitat Politècnica de Catalunya* (2015). URL: `http://security.nknu.edu.tw/crypto/papers/Evaluation%20of%20Cryptographic%20Capabilities%20for%20the%20Android%20Platform%202015.pdf`.

[Goo18] Google. *Android 8.1 Features and APIs*. 2018. URL: `https://developer.android.com/about/versions/oreo/android-8.1`.

[Kel14] Gordon Kelly. "Report: 97% of mobile malware is on Android". In: *Forbes Magazine* (2014).

[Mas16] Brian Mastroianni. "Survey: More Americans worried about data privacy than income". In: *CBS News* (2016).

[McR18] Hamish McRae. "Companies have been selling our data in exchange for 'free' products and services for a long time – Facebook's not so different". In: *Independent* (2018). URL: `https://www.independent.co.uk/voices/facebook-data-scandal-free-products-sheryl-sandberg-a8294006.html`.

[Mer17] Robert Merkel. "Why do computers, smart phones and other devices slow down over time?" In: *ABC* (2017). URL: `http://www.abc.net.au/news/2017-10-23/why-computers-and-smart-phones-slow-down-over-time/9070932`.

[Thi10] Joshua Thijssen. “Encryption operating modes: ECB vs CBC”. In: (2010). URL: https://adayinthelifeof.nl/2010/12/08/encryption-operating-modes-ecb-vs-cbc/.

[unk] Author unknown. “Benchmarks”. In: *CrytoPP* (). URL: https://www.cryptopp.com/wiki/Benchmarks.

[unk16] Author unknown. “Which Metals conduct heat best?” In: *Metals Supermarkets* (2016). URL: https://www.metalsupermarkets.com/which-metals-conduct-heat-best/.

[unk17] Author unknown. “Antutu Benchmark”. In: (2017). URL: http://www.antutu.com/en/index.htm.

[Wal10] Michael A. Walker. “Standard Method of Evaluating Cryptographic Capabilities and Efficiency for Devices with the Android Platform”. In: *Institute for software integrated systems, Vanderbilt University* (2010). URL: https://ptolemy.berkeley.edu/projects/truststc/education/reu/10/Papers/WalkerM_paper.pdf.