AseCrypto goes Mobile

Performance comparison

Project work 1 by Wolfgang Gaar, BSc

Table of Contents

Introduction

The „AseCrypto goes Mobile“ app has been developed as a part of the first project work. It is freely available on Github.com.[1] An example screenshot of the app can be looked at in Figure 1. The app has been developed using the new Android Studio from Google and may therefore directly imported and viewed using this development environment.

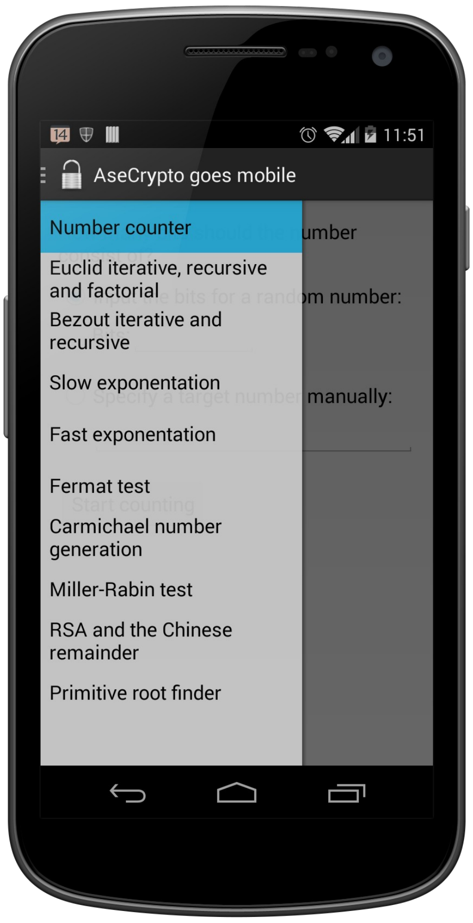

Illustration 1: Example screenshot of the developed app.

The second part of this project work is this performance comparison at hand. On the following pages, the performance of the exercises is compared between execution on a desktop computer and the developed AseCrypto app.

A performance comparison between desktop computers and mobile devices is interesting, as the mobile counterparts are much more limited in computing power and also do have other constraints upon them, such as remaining battery power, available memory, or etc.

The given task was to provide performance comparison for bit sizes so that all exercises are finished

1 AseCrypto goes Mobile app source code: https://github.com/nohum/asecrypto-goes-mobile-app

in a reasonable amount of time. However, there are some unforseen constraints too. For example, recursion is quite fragile on mobile devices and is therefore limited to prevent application crashes. If such countermeasures are taken, the reader is going to be informed about it.

Performance measurement

In this part of the paper, the performance of every single exercise is compared. This is done by using the same input value (usually bits) on both desktop and mobile and measuring the milliseconds it took to complete the exercise. All tests on the mobile device are done without the power being plugged in.

All the files that have been used to generate the charts also packaged with the application sourcecode and can be found in the „paper" subdirectory.

Regarding the test setup, the desktop computer is an ASUS N53Jq notebook utilizing an Intel CPU with 4 cores and a computing power of 1.6 Gigahertz.[2] The used mobile phone is called „Moto G" and produced by a vendor called Motorola. It too utilizes a quad-core CPU, but only has 1.2 Gigahertz of computing power.[3]

Please also note, that the test examples on the mobile device had to be implemented in a asyncronous manner using Android's AsyncTask. This is required by the Android UI framework to prevent the UI from blocking.[4]

Number counter

The first example is from the first lab session. Task was to implement a BigInteger number counter. As you can see in the chart located in figure 2, the desktop system is quite fast while the mobile device lacks speed. In terms of a reasonable calculation time, the mobile device did need 21 minutes to count a 26 digit bit number. This can be seen in the bar labelled „Mobile app with isCancelled()".

Regarding the statement of a reasonable amount of execution time for every exercise, the mobile device has clearly lost this time. Clearly, the time needed by the desktop is so small that the bar is not even visible compared to the mobile device's time data. Therefore, the test setup has only been done for up to 26 bits.

2 Specifications of the laptop computer: http://www.asus.com/Notebooks_Ultrabooks/N53Jq/specifications/
3 Specifications of the mobile device: http://www.motorola.com/us/moto-g-pdp-1/Moto-G/moto-g-pdp.html#moto-g-pdp-specifications
4 Regarding asyncronous processing of data, please check: http://developer.android.com/guide/components/processes-and-threads.html#Threads

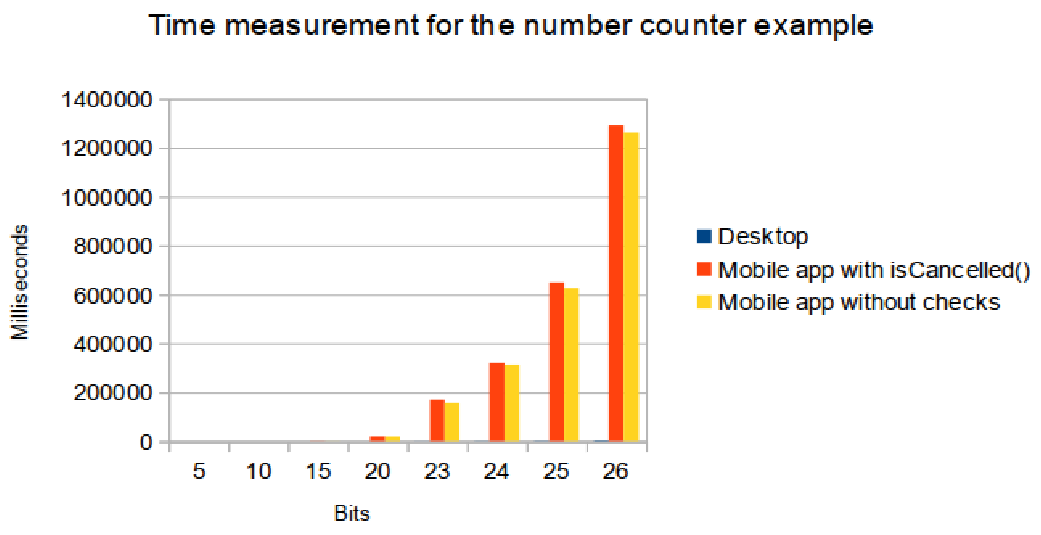

Illustration 2: Time measurement of the number counter example

Asyncronous overhead

Still, there is one aspect to note: The Android operating system expects such tasks to be asyncronous. As all examples have been implemented in such a way, Android requires that the developer checks if the asyncronous task has been cancelled using a „isCancelled()“ method call.

For the number counter example, this method call has been removed temporarily to show the overhead of this function call (which is called after every increment). As you can see, the time difference is not big and only ranges between 7 to 30 seconds. However, this time benefit does not mitigate the negative effects of removing the „isCancelled()“ call. Removing this call would remove the possibility of manually cancelling the task and also disturb the Android operating system in stopping tasks if the app has been closed or on other conditions.[5]

Euclidean algorithm

The Euclidean algorithm is seperated into three charts for the iterative, recursive and factorial approach. The bits describe the used bits for both numbers and is capped by 500 bits to make all three graphs comparable to each other. This maximum has been chosen because for the recursive approach, a maximum of 512 bits is possible, as otherwise the application would run in a stack overflow error and thus crashing the application on the mobile device.

5 isCancelled() documentation of the Android project: http://developer.android.com/reference/android/os/AsyncTask.html#isCancelled%28%29

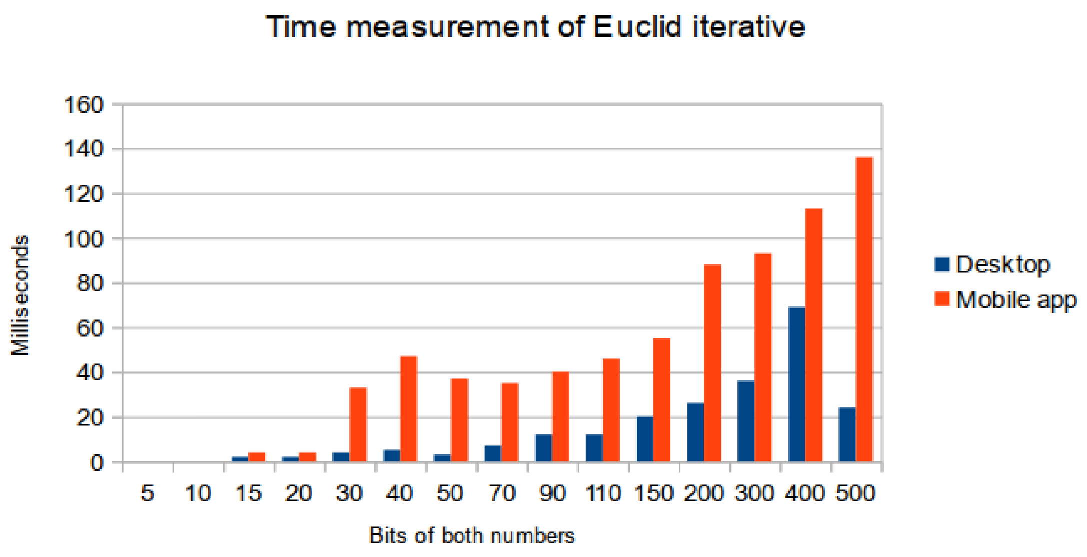

Illustration 3: Time measurement of Euclid iterative

The performance of the iterative version may be looked at in graph 3. As can be seen, the desktop outperformes the mobile device if the numbers start getting bigger than at about 20 bits.

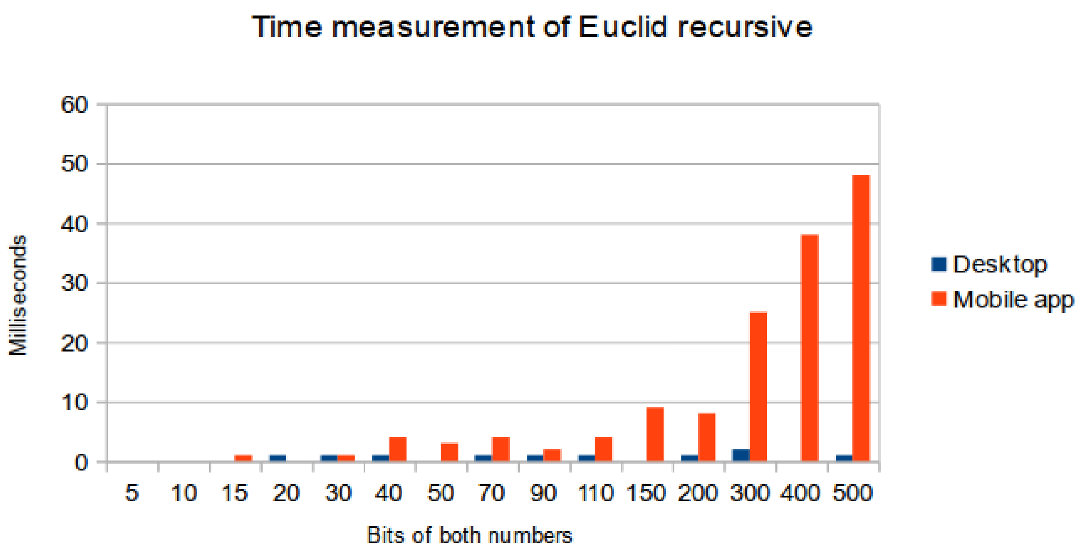

Illustration 4: Time measurement using Euclid in its recursive form

As visible in figure 4, both desktop and mobile applications manage to process the Euclidean algorithm very fast. While the desktop computer always resides in the one digit milliseconds scale, the time needed by the mobile app rapidly grows starting with 300 bits. Despite that, it has to be noted that the overall execution times still appear fast to the user.

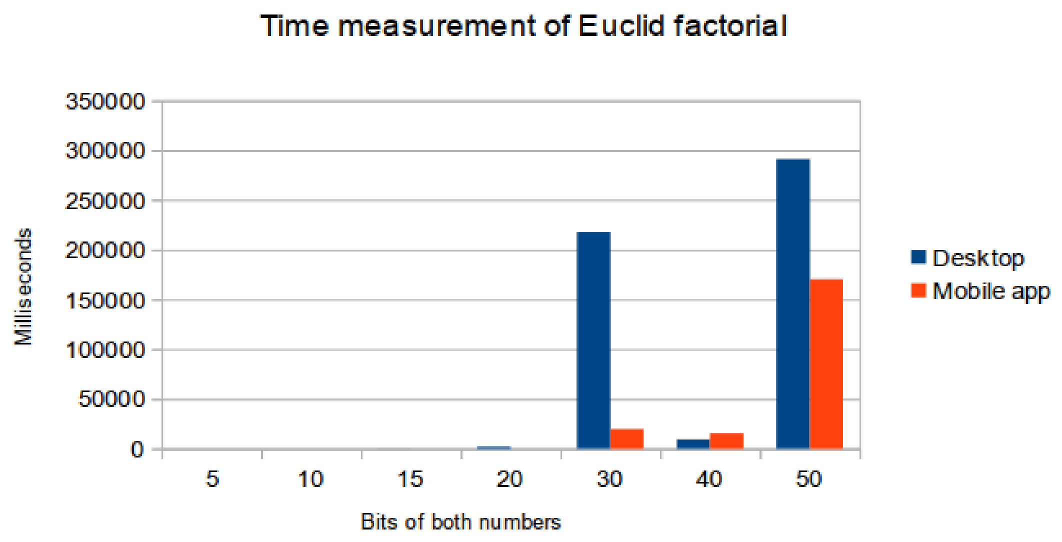

Illustration 5: Time measurement of the factorial Euclid

The factorial version of the Euclid, visible in figure 5, yielded quite interesting results. First at all, with more than 50 bits both desktop and mobile app calculated longer than 20 minutes without returning a result set. Therefore, this graph is limited to 50 bits. Additionally note that the execution of the factorial algorithm has been faster on the mobile device when utilizing calculations with 40 or 50 bits.

Extended Euclidean algorithm

The Bezout algorithm as well needed a cap in the possible bit size. The maximum bit size for the mobile app has been limited to 285 bits. As usual, the bit size is entered for both numbers.

As visible in figure 6 the iterative Bezout algorithm is quite stable in regards of milliseconds on the desktop computer. Interestingly enough, with low bit sizes even the mobile app is faster than the desktop counterpart. Only with growing bit size the mobile app decreases in speed and needs longer to accomplish its task.

Using the recursive implementation, both test environments can compete against each other until the bit size hits 30 bits. After that, as to be seen in figure 7, the mobile device always needs longer to finish its calculation.

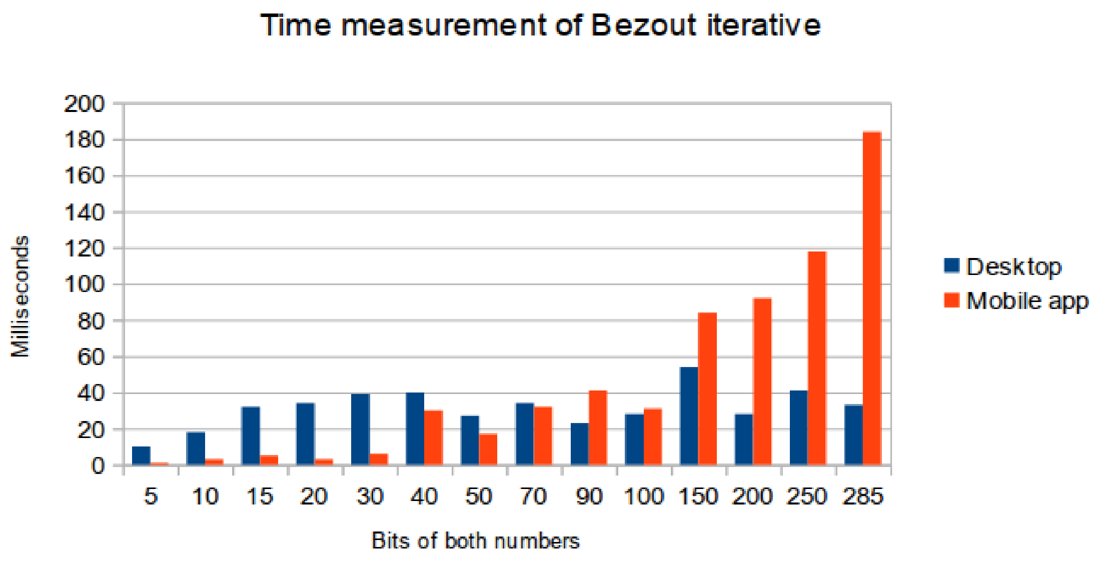

Illustration 6: Time measurement of the iterative Bezout implementation

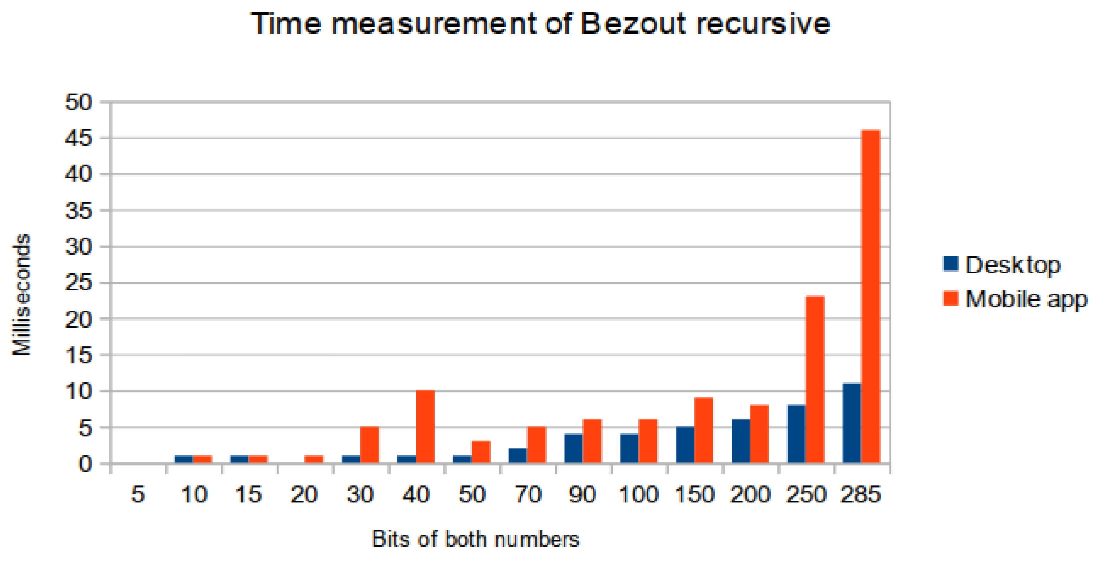

Illustration 7: Time measurement of the recursive Bezout

Slow exponentiation

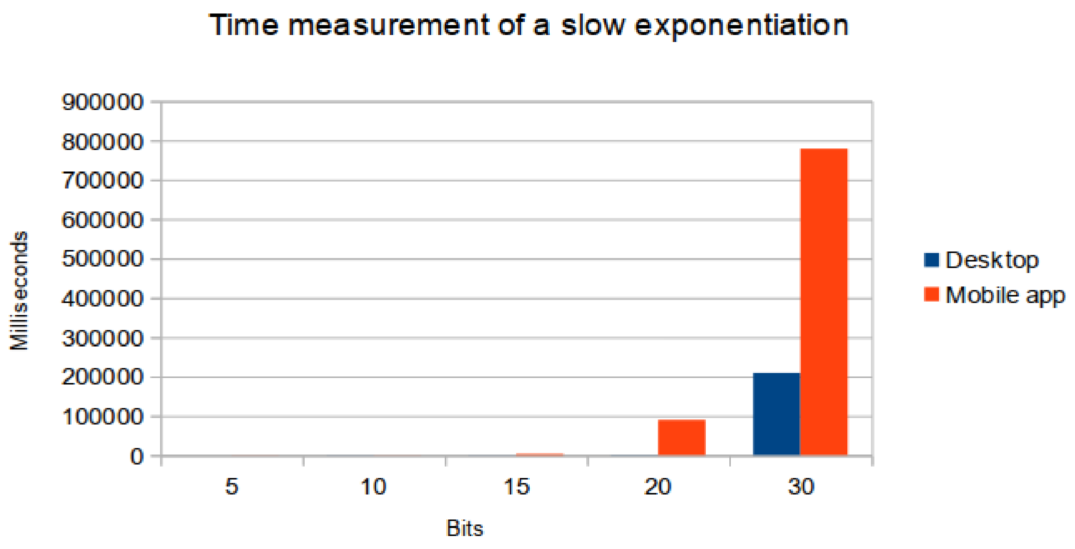

Illustration 8: Time measurement of a slow exponentiation

The slow exponentiation did take long starting with 30 bits, therefore the dataset has been limited to 30 bits. This can be seen in figure 8.

Fast exponentiation

Contrary and as the name implies, this exponentiation method did beform better as the previous one. The results are viewable in graph 9.

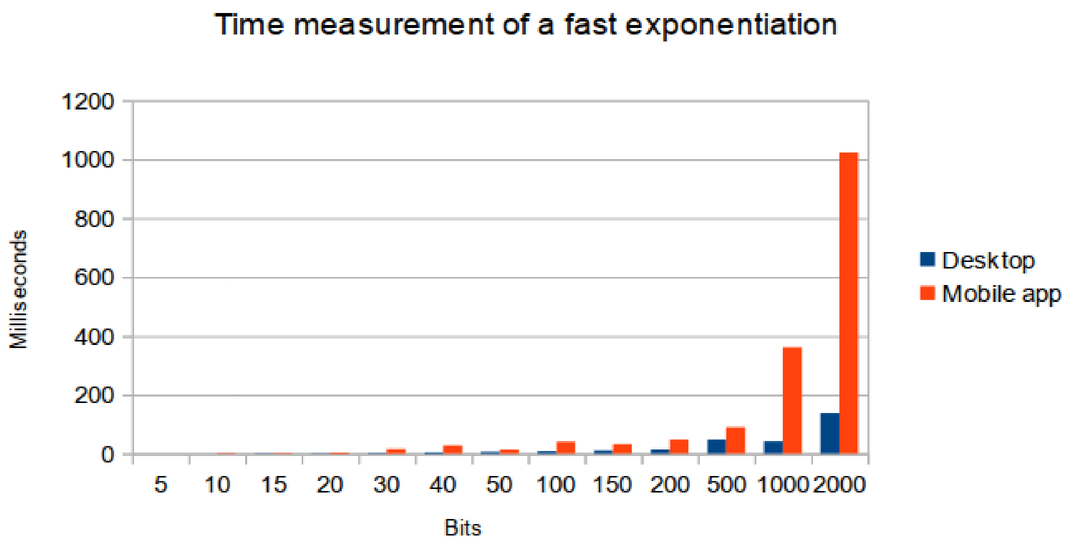

Illustration 9: Time measurement of a fast exponentiation

As visible, the desktop computer is quite faster than the mobile device. Starting with 2000 bits, the calculation did take some time on the mobile device. For such a calculation, this was chosen as the maximum of time that should be acceptable.

Fermat test

The fermat test is executed using a known prime, e.g. the number eleven. After that, for a given number of test runs, the taken time is measured. The modified Fermat test for the desktop computer can be found in the „paper/perf-fermat.zip“ archive.

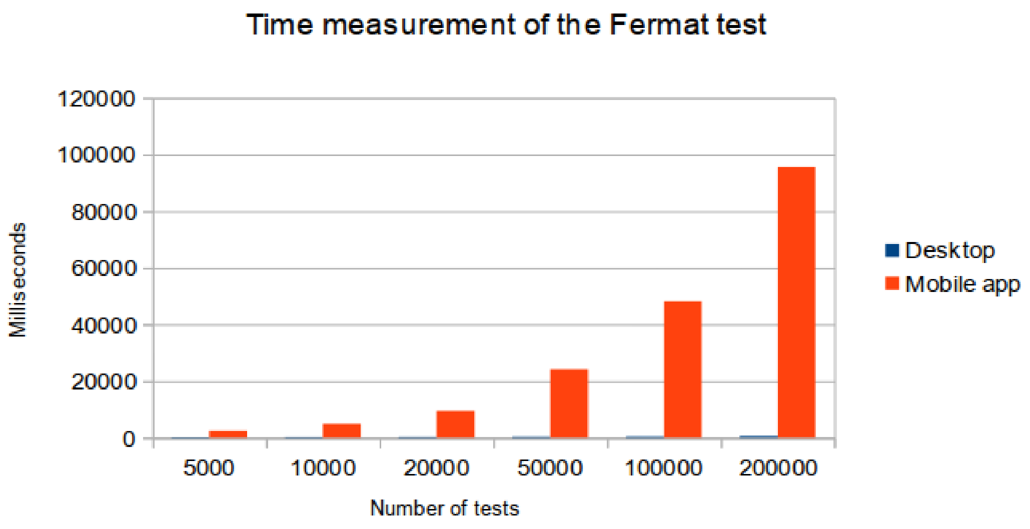

Illustration 10: Time measurement of the Fermat test

As can be seen in figure 10, the fermat test is significantly faster on the desktop application.

Carmichael number generation

The Carmichael number generation is done on the desktop using the java application located in the „paper/perf-carmichael.zip“ archive. As can be seen in illustration 11, the desktop application did outperform the mobile app by any means.

The maximum bit size has been fixed with 200 bits as any higher bit sizes like e.g. 500 bits already did take too long to calculate on the desktop.

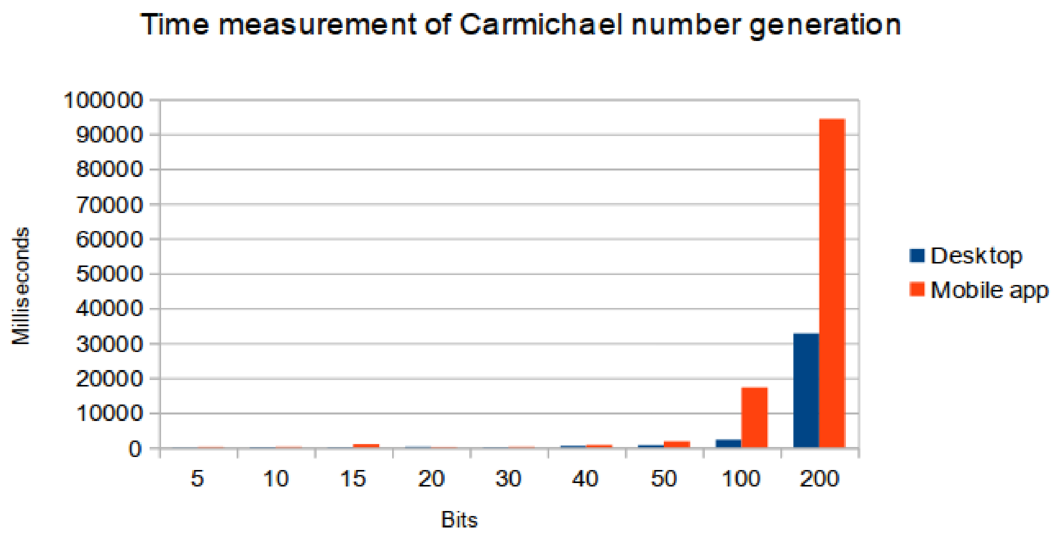

Illustration 11: Time measurement of Carmichael number generation

Miller-Rabin test

The Miller-Rabin test is done on the desktop using the java application that is located in the „paper/perf-millerrabin.zip“ archive. The desktop application also outperforms the mobile application by any means. As viewable in figure 12, a maximum of 50.000 test runs has been done because of the long time it did take the mobile device to finish.

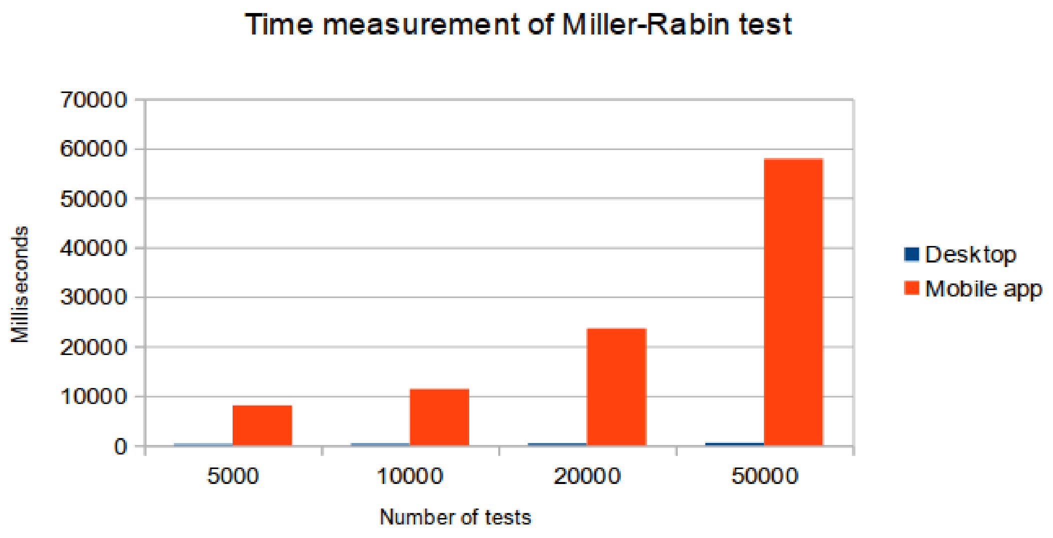

Illustration 12: Time measurement of the Miller-Rabin test

RSA decryption

The example application for the desktop platform can be found in the „paper/perf-rsadecryption.zip“ archive. The p and q values have a bit length 16 bits. The time measurements of the RSA decryption can be seen in figure 13. The Chinese remainder theorem greatly speeds up the decryption on the mobile app.

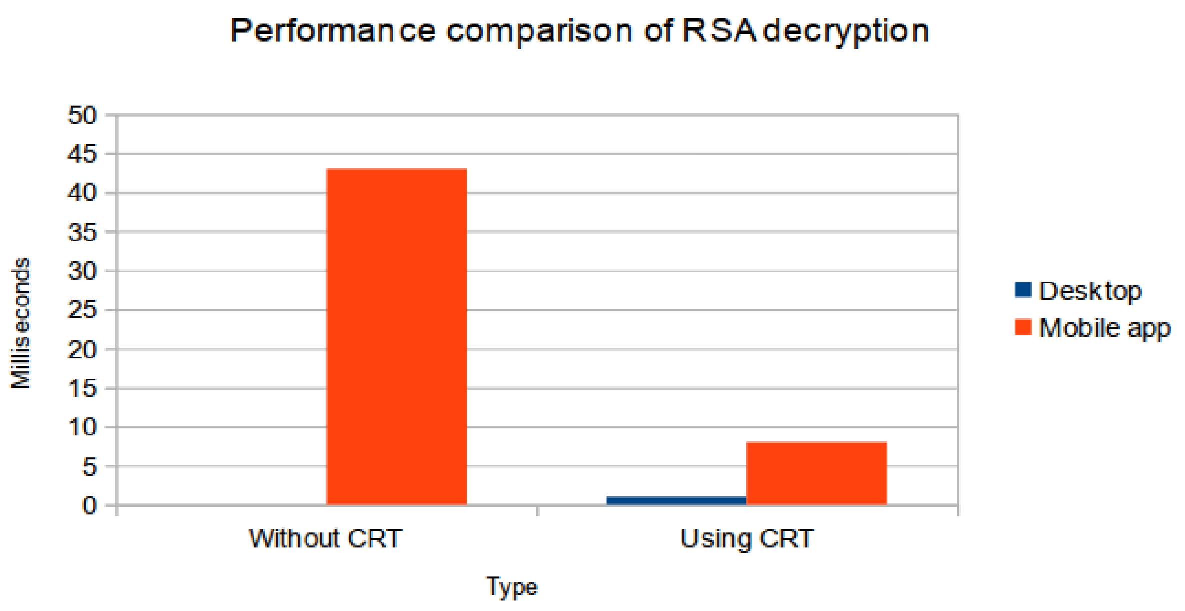

Illustration 13: Performance comparison of RSA decryption methods

Primitive root finder

Finding primitive roots on the desktop and mobile device did not yield a clear result of whom is performing faster. Clearly, as viewable in illustration 14, the mobile device needs longer in terms of time if the bit size gets too big. However, with small bit sizes, even the mobile app is sometimes faster than the desktop application.

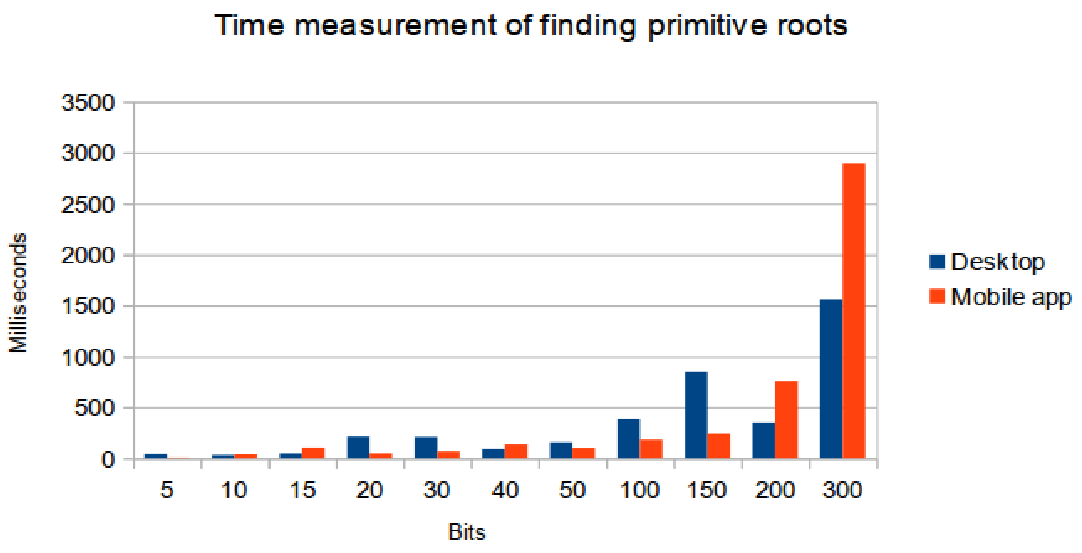

Illustration 14: Time measurement of finding primitive roots

Conclusion

As seen, the performance is generally better on the desktop platform. In some rare cases, the mobile device outperforms the desktop. However, there are some additional constraints on the implementation for the mobile phone. For example, for an Android application, all calculations must be done asyncronously while also checking if the calculation must be aborted. It lies in the nature of desktop computers that these checks or measures are not a necessity on desktop platforms as these do not have to fight e.g. battery drain the same like mobile devices.