IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received October 27, 2017, accepted November 22, 2017, date of publication December 4, 2017,
date of current version February 14, 2018.

Digital Object Identifier 10.1109/ACCESS.2017.2779131

What Do Software Developers Need to Know to Build Secure Energy-Efficient Android Applications?

JOSÉ A. MONTENEGRO, MÓNICA PINTO, AND LIDIA FUENTES
CAOSD Group, Andalucía Tech, Universidad de Málaga, 29071 Málaga, Spain

Corresponding author: Mónica Pinto (pinto@lcc.uma.es)

This work was supported by FEDER funds under Project Magic P12-TIC1814 and Project HADAS TIN2015-64841-R.

ABSTRACT Green computing is a growing trend in computing, pursuing the goal of helping software developers to be more aware and produce energy-efficient software. This is especially relevant for battery-powered mobile applications, where a minimal energy consumption is desired to both mitigate the greenhouse effect and extend the battery lifetime. In this paper, we analyze the energy consumption and execution time of cryptographic primitives in Android devices. Our ultimate goal is to help Android application developers, especially those who are not experts in security, to choose the most energy-efficient cryptographic algorithms considering different security providers and security transformations. Information to make a tradeoff between energy and time consumption is also provided, being especially useful when the differences in energy consumption of different alternatives are not so significant. We have conducted our experiments with an energy profiling tool based on the PowerTutor application, which has been adapted to automate the energy profiling. Our results show that this type of power consumption studies is necessary, because selecting the most energy-efficient configuration depends on many factors, and some of the choices are not obvious to developers.

INDEX TERMS Cryptographic primitives, Android, energy consumption, execution time, energy-efficient configurations.

I. INTRODUCTION

The percentage of global emissions attributable to Information Systems is expected to further increase in the coming years, due to a proliferation of Internet-connected mobile appliances (e.g., mobile phones or tablets). As a result, the development of applications for battery-powered mobile devices should consider minimizing the energy consumption to be of the highest priority to both mitigate the greenhouse effect and extend the battery lifetime. However, developers are not always aware of the energy consumed by their software solutions [1], [2]. Trying to address this issue, Energy-aware software development (or Green Computing [3]) is a growing trend in computing, pursuing the goal of helping software developers be more aware of the energy-efficiency of software [4]–[7].

People principally use mobile devices to send and receive more or less private information, which makes security an important concern in the development of mobile applications. Cryptographic algorithms are used to provide authentication, confidentiality and integrity, providing different security levels. Traditionally, the implementation of cryptographic primitives requires significant energy to process data, and some recent studies show that this is also true for mobile phones [8], [9]. Therefore, software developers should be aware of the impact on power consumption using cryptographic algorithms in their applications has.

Recently, several experimental studies have been conducted to investigate the energy consumption of concrete implementations of cryptographic algorithms and security protocols for different devices [8], [10]. However, they focus on other aspects of security (e.g., secure communications) or try to answer different questions. In many cases, the main goal of these studies is to compare the power consumption of different cryptographic primitives of a concrete implementation (i.e., a concrete security provider or API), neglecting the point of view of the developer who may be interested in exploring the use of alternative cryptographic providers. Indeed, these studies do not cover all the security alternatives from different viewpoints, i.e., different cipher modes and padding, specific default configurations for each provider and different security levels, with many open research questions remaining.

In this paper we focus on studying the energy consumption of cryptographic primitives from the point of view of the application developer. We have tailored this study to Android devices, the most popular operating system of mobile devices in recent years. So, our goal is to help Android application developers to choose the most energy-efficient cryptographic algorithms, considering different implementation libraries (i.e., cryptographic providers). The target audience of the experimental study presented in this paper is a software developer, non-expert in security issues, specifically with poor knowledge of crypto algorithms' internals and who is interested in knowing the energy consumed by alternative security providers under different conditions.

We have applied our experiments to three different providers (IAIK, BC and SC), and several different primitives (encrypt, decrypt, sign, verify and mac) and the most used cryptographic algorithms for each primitive. In addition the different combinations of the primitives' parameters have been considered (e.g., different key sizes, modes and paddings for the cipher algorithms). In this paper we intent to provide empirical evidence to answer specific questions that developers concerned about power consumption should ask. Considering Android devices and different crypto providers, our research questions try to give developers recommendations as to which crypto provider to use, or to explore the relationship between security and energy consumption. In addition, considering software developers tend to use the default security transformation offered by providers (i.e., the default configurations), we have measured the energy consumption of these transformations to see the cost of incrementing the security in each crypto provider, in terms of energy. The purpose of this is to be able to advise Android developers on replacing the default mode with more secure ones where the energy cost is not too high. Although assessing power consumption is our primary aim, execution time is also measured and used in our analysis to make a tradeoff between energy and time when differences in energy consumption are not so significant.

We have used an energy profiling tool that is based on the PowerTutor [11] application. Section III explains our motivation behind using a software tool. Energy profiling tools are the preferred solution since they allow finer-grained measurements and also because the experiments can be reproduced, which, with hardware solutions is not always possible. PowerTutor has been adapted to automate the profiling procedure, but without modifying the energy model procedures. The application uses a theoretical energy model built for the phone models HTC G1, HTC G2 and Nexus One. Due to the large amount of information generated, in this paper we only show the data from our experiments for the Nexus One. However, a HTC G1 has also been used to validate the results. Both phones provide similar results. Note that the PowerTutor application, in addition to many other profiling mechanisms [12]–[15], provides an estimation of the power consumption. However, this is not a limitation because, for our study the exact number of milliwatts that are consumed by a certain configuration is not relevant. Rather, we are interested in establishing comparisons between the energy consumed by different configurations. Note, that considering that a high percentage of mobile applications use cryptographic algorithms [16], and that they are executed in millions of devices, a small energy saving in a single device will result in huge global energy saving.

Following this introduction the paper is organized as follows. Section II describes the energy profiling tool used to perform our experiments. Then, our experimental setup, discussing first the high variability of cryptographic providers, primitives and configuration parameters and then selecting the ones considered in our study is described in Section III. The experimental results are presented and analyzed in Section IV, and the threats to validity are discussed in Section V. Finally, Section VI presents the related work and Section VII the conclusions and future work.

II. RELATED WORK

Energy consumption has become a central issue for mobile applications. Applications providing similar services can have different energy consumption and both software developers and mobile users need to be aware of these differences. The number of papers that focus on the generation of energy profiles of different characteristics of mobile applications has considerably increased in the last few years. Some of them are [8]–[10] and [17]–[22]. Other papers focus on analyzing the energy consumption of real market Android applications [23] or on defining mobile applications' development guidelines for energy [24]. As we do in this paper, the final goal of these proposals is providing guidelines to software developers, so that they can develop more energy-efficient applications. Additionally, based on such results, which analyse the energy consumption of applications, it is possible to improve the code to reduce the energy consumption [25]. Although many of these proposals [9], [17]–[20], [22], [23] are for the Android platform, not much attention has been paid to the energy consumption of Android cryptographic primitives, with the exception of the work presented in [26] and [27].

Thus, in this section we focus on those proposals that generate energy profiles for cryptographic primitives [8], [10], [21], [26], [27], and especially [26], [27] which are the closest to our approach. Many of these proposals focus on generating energy profiles for mobile applications [8], [10], [21], [26], [27], but only [26], [27] consider the Android operating system. All these papers calculate and compare the energy consumption of different cryptographic primitives, varying different parameters, such as the key sizes and the operations. In some of them the ultimate goal is to move beyond the mere comparison of levels of energy consumption. For instance, the work in [8] focuses on encryption in secure communications and the work in [21] analyzes the consumption of cryptographic primitives in the context of the SSL protocol. However, only the work in [8] takes into consideration that the algorithms can work in different modes (e.g. ECB, CBC, CFB, OFB, CTR and CCM). The rest of the proposals do not give any information about the mode they are using. Moreover, none of them consider different paddings or different security providers and they do not indicate the ones they are using in their experiments. The default modes selected by the providers are not considered either. Since the provider, the mode and the padding can influence the security level it is impossible to really know the security level that these proposals are analyzing. Moreover, since the provider, the mode and the padding that are used in the experiments are unknown, the experiments cannot be reproduced and the results of different proposals are not comparable with each other. The main differences between these approaches and our work is that our analysis takes into consideration all the parameters that may influence enegy consumption of cryptographic primitives—i.e., different crypto providers, different algorithms and operations, different modes and different paddings. Concretely, we focus on those transformations (algorithm/mode/padding) that are the default ones selected by the providers and those that are the recommended ones by security standards. An exception is the work in [26], where the execution time, voltage and memory consumption are compared for the AES and the PRESENT cryptographic algorithms in Android. There are however important differences with our work. Firstly, in this paper only two cryptographic algorithms are considered so the scope of the study is very limited. Secondly, the authors carry out the experiments using a homemade implementation of the algorithms, rather than widely used implementations of security energy providers. This makes the comparison of the energy consumption of existing security providers impossible. Moreover, only one key size is considered for each algorithm. Finally, the modes and paddings of the security transformations are not taken into account in the experiments. In conclusion, the results obtained in this paper cannot be easily reused by other Android software developers. Similar work is presented in [27], where the authors investigate the energy spent by the most popular crytographic algorithms. They focus on cyber security and their results indicate that it is possible to develop a more green and secure Internet. They consider the same classes of algorithms as in our approach (encryption, hashing and message authentication), with different key sizes and different traffic loads over the network, but they do not consider different modes and padding for the encryption algorithms or different providers (only the BC provider is considered). The most important difference is that they do not really measure energy consumption as we do. Instead, they just exploit the relationship between the CPU usage and the consumed power.

An approach with similar goals to ours, but focusing only on performance evaluation instead of energy and time-consumption evaluation, is [28]. In this paper the authors consider different cryptographic providers for Android, including those considered in our study, and evaluate the performance of different algorithms taking into consideration not only the key length and the different primitives, but also the mode and the padding selected in the transformations. In our study, we have taken the algorithms' requirements for developing secure applications from this paper, since they were recopilated based on the information available in the different security standards. There are other publications that contain similar performance studies of cryptographic primitives but focusing on wearable Android devices [29] or IoT platformas and operating systems [30].

III. ENERGY PROFILING TOOL

Several proposals exist for estimating the energy consumption of mobile applications in Android [11]–[15]. As stated in [5], the main differences lie in whether they are hardware or software-based and in the measurement granularity (application, process, thread or method). Many of them [11]–[13] are based on the definition of a power model based on direct measurements in specific devices. These models generally provide quite precise estimations, although they are built for just a few devices. Others [31] use the power information provided by a mobile processor and thus applies to all devices with that processor. Since our analysis is based on the comparison of the energy consumption of different combinations of provider/operation/algorithm/parameters, and thus we are not concerned with the absolute consumption values, our approach should provide similar results with any of these tools.

Unfortunately, there are few trustworthy energy profiling tools for Android. We first tried with the Trepn Profiler [31] application, because power profiling is available for all devices with a Qualcomm Snapdragon processor. However, energy profiling at the method level does not work properly in Trepn Profiler for short duration methods, as is our case. A second one is required to register the measurements and the execution of most of the methods we are testing lasts less than a second in the latest processors. Some approaches use a hardware solution, but this has two main problems: (1) it is very difficult to "estimate" which part of the code is responsible for the energy consumption; and, (2) the reproducibility of the experiments is seriously compromised. Thus, our energy profiling tool is based on the PowerTutor[1] project, described in [11]. Basically, PowerTutor is an Android application for profiling energy consumption based on the CPU, network interface, display, and GPS elements. The application uses a theoretical energy model built exclusively for three phone models: HTC G1, HTC G2 and Nexus One. These phone models have a maximum of 2 cores and run at lower clock rates than modern devices. However, this is not a limitation for our study because it is based on the comparison of the relative power consumption of different cryptographic primitives and providers running on exactly the same device. The

[1]http://ziyang.eecs.umich.edu/projects/powertutor/documentation.html

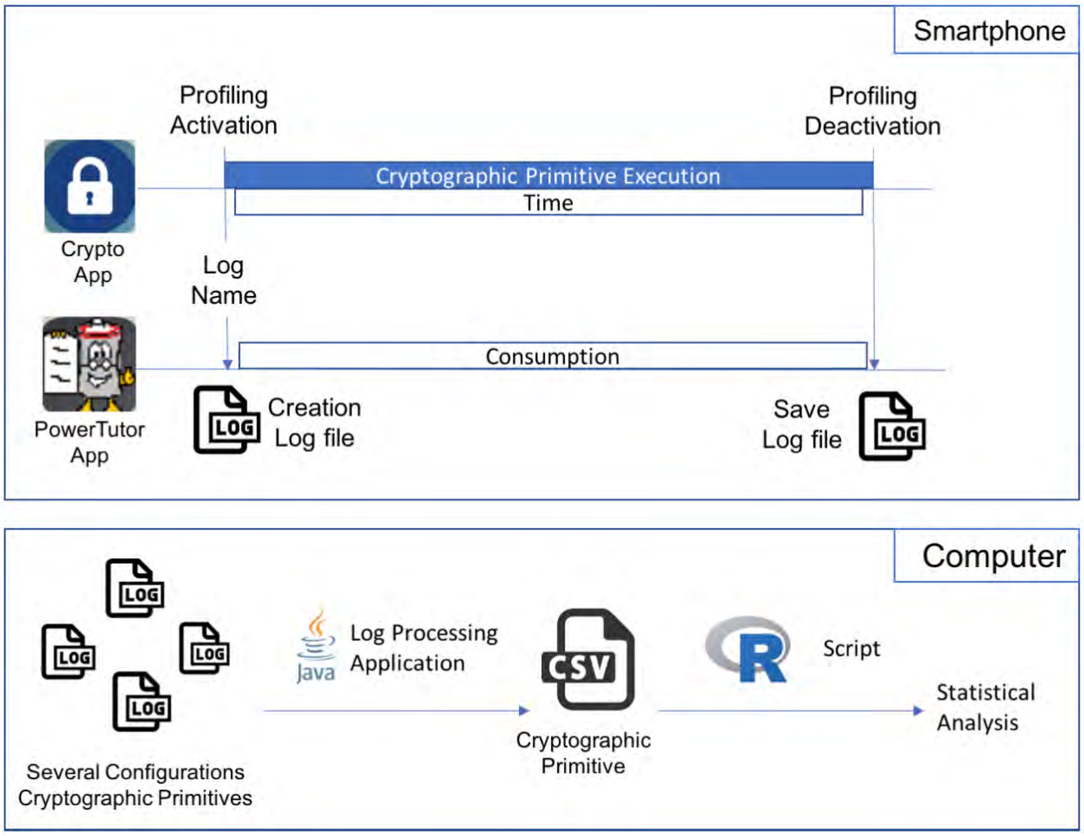

FIGURE 1. Profling platform designed to collect consumption information.

absolute values are not relevant for our study. What is more important, we have tested the latest versions of the cryptographic primitives implemented by the providers that are used in the most recent Android versions, so we have tested the latest cryptographic software currently used in modern devices.

The original PowerTutor application was designed to visualize the consumption information, but it requires user intervention to store the information in a log file. Thus, the application is very useful to measure the power information of our cryptographic primitives, but it was not an easy task to automate the profiling procedure. In order to automate the procedure, several changes were included in the application, without any change to the energy model procedures.

The profiling platform designed to collect energy consumption information in our experimental study is shown in Figure 1. Firstly, we modified the PowerTutor application to include an Android mechanism for automatically collecting the energy consumption information. Moreover, a Crypto application has been deployed with all the cryptographic primitives detailed in Section IV-C. The goal of this application is twofold, first to execute the cryptographic primitives and second to measure the execution time of these primitives, as shown in Figure 1. The execution time is also measured for the Crypto application. The primitive to be evaluated is coded between two commands that activate and deactivate the profiling service of the PowerTutor application. The activation process sends a string that represents the cryptographic primitive to be evaluated and contains the name of the log file that will be stored automatically when the profiling service is deactivated. In this way, the PowerTutor application is used without user interaction and the evaluation can be automatized. The generated logs are then processed on a destktop computer to provide the information in a format that can subsequently be analyzed by software developers. We have used the R language for the statistical analysis. A Java program and R scripts have been deployed to process the log files. The Java program processes several log files to create a CSV file, for each primitive and provider, which contains energy and time values. After that, CSV files have been analyzed using the R language. The output of R scripts has been used to perform the comparative analysis in the paper. The modified PowerTutor and the Crypto applications can be downloaded at the following url: http://www.lcc.uma.es/~monte/CryptoConsumption/

IV. EXPERIMENTAL SETUP

In this section we describe the variables that are taken into consideration in our experimental study. Firstly, we describe the selected cryptographic providers. Then, the available Android cryptographic primitives are described, and the reasons for selecting the ones used in our study are provided. Finally, the default modes used by each provider are detailed. Later we use these default modes to advise developers to replace the default mode with more secure ones where the energy cost is not too high.

A. CRYPTOGRAPHIC PROVIDERS

The Java Cryptography Extension (JCE) [32] is an API that provides a uniform framework for the implementation of security features. There are different Cryptographic Service Providers that supply specific implementations of a subset of cryptographic services. Three cryptographic providers for JCE have been taken into consideration in this paper, BouncyCastle (BC),[2] SpongyCastle (SC),[3] and Institute for Applied Information Processing and Communication (IAIK).[4]

BouncyCastle is an independent cryptographic provider that can be executed on any Java platform. Android has included a reduced and adapted version of the BouncyCastle library in the system, therefore Android users have access to cryptographic primitives without installing an additional library. SpongyCastle is also an adaptation of the original BouncyCastle library for the Android platform. SpongyCastle is not Android distribution dependent and includes the implementation of most well-known cryptographic primitives. At first glance, SpongyCastle and BouncyCastle include the implementation of the same algorithms. Thus, SpongyCastle has been included in our study to verify whether the modifications of BouncyCastle performed by Android have a positive or negative influence on the power consumption. IAIK is a library deployed by the Graz University of Technology. IAIK is a well-known cryptographic provider, and can be used with an educational license. IAIK has been optimized for speed and memory consumption and thus has been included in our study to verify if it is greener than other providers.

The latest available versions at the moment of the experiments have been choosen for the three providers. These versions run in the latest versions of the Android platform.

B. ANDROID CRYPTOGRAPHY PRIMITIVES

Android documentation classifies the cryptographic primitives into three categories: MAC (Message Authentication Code), Cipher and Sign.

A MAC algorithm is a symmetric key cryptographic technique to provide integrity and message authentication, i.e. to identify the originator of a message. Both the sender and the receiver share a symmetric key. Table 1 details the MAC algorithms available in Android.[5] The algorithms in grey were discarded from our experimental study for various reasons, which are indicated as table footnotes and also discussed in the next subsection.

TABLE 1. MAC algorithms available in Android.

Name	Key Size		API Levels
	IAIK	BC/SC	
HmacMD5	512	128	1+
HmacSHA1	512	160	1+
HmacSHA224[1]	-	-	1–8, 22+
HmacSHA256	512	256	1+
HmacSHA384	1024	384	1+
HmacSHA512	1024	512	1+
PBEwithHmacSHA[2]	-	-	1+
PBEwithHmacSHA1[2]	-	-	1+

[1] API label 10 not provided (required by PowerTutor)
[2] Not recommended by security standards

TABLE 2. Sign algorithms available in Android.

Algorithm	Hash	Key Sizes	API Levels
DSA	NONE	1024, 2048[3]	1+
	SHA1		1+
DSS[2]	NONE	-	1–19
ECDSA[1]	NONE	-	11+
	SHA1	-	11+
	SHA256	-	11+
	SHA384	-	11+
	SHA512	-	11+
RSA	NONE[1]		17+
	MD5	512, 1024, 2048	1+
	SHA1		1+
	SHA256		1+
	SHA384		1+
	SHA512		1+

[1] API label 10 not provided (required by PowerTutor)
[2] Not recommended by security standards
[3] BC does not support 2048 bits length

A sign algorithm is used to create digital signatures so that a person or entity can be bound to the digital data. The receiver can then verify the authenticity of the signature. The digital signature scheme is based on public key cryptography, using a public-private key pair. Table 2 details sign algorithms available in Android.[6] The algorithms discarded from our study are marked in grey (see Table 2 footnotes for explanation). Key sizes are only specified for the included algorithms.

A cipher algorithm is used to encrypt and decrypt information. A distinction is made between symmetric key encryption algorithms, where the same keys are used for encrypting and decrypting the information; and asymmetric key encryption algorithms, where different keys are used. In addition, cipher

[2]https://www.bouncycastle.org/
[3]https://rtyley.github.io/spongycastle/
[4]http://jcewww.iaik.tu-graz.ac.at/
[5]https://developer.android.com/reference/javax/crypto/Mac.html
[6]https://developer.android.com/reference/java/security/Signature.html

TABLE 3. Cipher algorithms available in Android.

Cipher	Mode	Padding	Key Size[1]	API Level
AES	CBC	ISO10126Padding NoPadding PKCS5Padding	128, 192, 256	1+
BLOWFISH[2]	CFB		-	10+
DES[2]	CTR		-	1+
DESede	CTS		112, 168	1+
PBEwithMD5andDES[2]	ECB		-	1+
PBEwithSHA1andDESede[2]	OFB		-	1+
RC4[2]	ECB	NoPadding	-	10+
RSA	ECB None	NoPadding	512, 1024, 2048	1+
		OAEPPadding		1+
		OAEPwithSHA-1 MGF1Padding		10+
		OAEPwithSHA-256 MGF1Padding		10+
		PKCS1Padding		1+

[1] Only provided for the algoritms used in our experimentation
[2] Not recommended by security standards

algorithms support different modes and padding. Ciphers process data blocks of fixed size. However, the message size is usually larger than the block size. Hence, the message has to be divided into a series of sequential message blocks. The cipher mode determines the way in which the cipher operates on these blocks and has a significant influence on the security level provided by the algorithm. The padding is a technique that adds bits to the message to change its length, either to obtain a fixed length required by the algorithms, or to prevent length extension attacks. Table 3 details cipher algorithms available in Android,[7] including the modes and padding supported by them. The key sizes are specified only for the algorithms included in our study.

C. CRYPTOGRAPHY PRIMITIVES SELECTED

The PowerTutor application requires the use of specific smartphone models; in our case, the phone chosen is a Nexus One (Android OS, v2.3.6, Qualcomm QSD8250 Snapdragon S1, 1.0 GHz Scorpion, 512 MB RAM, Removable Li-Ion 1400 mAh). The Android API level 10 is the maximum API level in these smartphones. Therefore, only the primitives available in API level 10 can be used.

Due to the high variability of cryptographic primitives we have narrowed the selected primitives to those that allow software developers to maximize the security levels of their applications, according to the information provided by security experts and security standards [33]–[36]. Default configurations used by the cryptographic providers are also considered.

The National Institute of Standards and Technology (NIST)[8] approved two block cipher algorithms from the cipher primitives in Table 3, therefore only the *AES* (Advanced Encryption Standard) and the *DESede* (Triple-Data Encryption Standard) algorithms are included. In the case of digital signatures NIST[9] approved three algorithms *RSA* (Rivest-Shamir-Adleman), *DSA* (Digital Signature Algorithm) and *ECDSA* (Elliptic Curve DSA). As shown in Table 2, *ECDSA* is not available in API 10, therefore only the *RSA* and the *DSA* algorithms are selected for the sign primitive. For generating and verifying message/data authentication codes there is just one algorithm HMAC[10] (Hash-based MAC). This algorithm is based on the cryptographic hash function [33], and in this case, NIST approved the following Secure Hash Algorithms (SHA) to be included in the Android platform: SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512.[11] Although MD5 (Message-Digest 5) is no longer recommended by NIST, we take it into consideration as it is still widely used.

In summary, ten primitives have been selected to evaluate three providers in a Nexus One smartphone. For the selected primitives, the key lengths are specified in Tables 1, 2 and 3. Several operations have been considered for each primitive. In MAC primitives we consider the operations MAC and Key Generation. In the case of sign primitives, the operations available for evaluation are Sign, Verify, and Key Generation. Finally, in each cipher primitive the operations are Encryption, Decryption and Key Generation. The first execution of the application described in Section III generates the bundle of keys necessary for cryptographic primitives to perform the encryption/decryption and sign/verify operations. The goal of the key generation and store is twofold; all the operations will be carried out, evaluated and compared with the same keys, and we save a substantial amount of time in key generation in each execution. The encrypted and signed messages are also stored so that the decryption and verify operations can be executed with the same values for all the providers.

[7] https://developer.android.com/reference/javax/crypto/Cipher.html
[8] http://csrc.nist.gov/groups/ST/toolkit/block_ciphers.html
[9] http://csrc.nist.gov/groups/ST/toolkit/digital_signatures.html
[10] http://csrc.nist.gov/groups/ST/toolkit/message_auth.html
[11] http://csrc.nist.gov/groups/ST/toolkit/secure_hashing.html

D. DEFAULT MODES

When the JCE API is used, java programmers must choose a correct *transformation* for each primitive. For example, for the cipher primitive, first the programmer selects a cipher algorithm, e.g., the AES algorithm. Then, according to the information shown in Table 3, the programmer selects one mode operation and one padding option. The triplet algorithm/mode/padding will define the transformation selected. For instance, if a programmer wants to cipher with the *AES cipher* in *CBC (Cipher Block Chaining) mode* and *PKCS5Padding* (Password-Based Cryptography Specification), this is done with the following Java code:

$$Cipher\ c = Cipher.getInstance(\text{``}AES/CBC/PKCS5Padding\text{''});$$

Android developers who are not expert in security can easily get to know about the cryptographic primitives available, i.e., the AES algorithm to cipher, but they will probably have little knowledge about the mode and padding variables [16]. Moreover, due to the large number of possibilities to choose from (see Table 3) and the difficulty to know which ones are the most secure, in many cases they just omit these variables and instantiate the algorithms with the default values, which are selected by the providers. This is possible because the JCE API allows programmers to get a default transformation for the selected algorithm with the following Java code:

$$Cipherc = Cipher.getInstance(\text{``}AES\text{''});$$

The default configurations for the cryptographic primitives selected in our study are shown in Tables 4 and 5. However, these default configurations are not necessarily the most secure ones, and this is because we are interested in considering these default configurations in our experimental study. In [16] the authors analyzed 11748 applications available from Google Play. In 5656 of these applications, developers only specified the cipher algorithm to perform encrypt or decrypt operations, omitting the mode and the padding. In consequence, Java determines the default transformation, i.e., the ECB (Electronic CodeBook) transformation, which is not considered secure. Therefore, 50% of analyzed applications have a low security level because developers are not aware of the security level of default transformation configurations or they do not know how to configure security transformations. Thus, it is useful to measure the energy consumed by the default configurations used by most non-expert Android developers to see if they are making good decisions from the point of view of energy consumption. We also want to analyze whether or not it is possible to use other transformations that provide a higher level of security without incurring a large increase in energy consumption. Note that it was not straightforward for us to know which were the default modes for each provider. We had to extract this information from the data obtained in our experimental results. This shows that software developers who decide to use the default modes do not really know the values that they are using for the mode and padding variables.

TABLE 4. IAIK default modes.

IAIK			
		Key Size	**Transformation**
Cipher	**AES**	128 bits	AES/ECB/PKCS5Padding
	DESede	168 bits	DESede/ECB/PKCS5Padding
	RSA	1024 bits	RSA/ECB/PKCS1Padding
Sign	**RSA**	1024 bits	NONEwithRSA
	DSA	1024 bits	–

TABLE 5. BC & SC default modes.

BC & SC			
		Key Size	**Transformation**
Cipher	**AES**	192 bits	AES/ECB/PKCS5Padding
	DESede	168 bits	DESede/ECB/PKCS1Padding
	RSA	2048 bits	RSA/NONE/NoPadding
Sign	**RSA**	2048 bits	NONEwithRSA
	DSA	1024 bits	–

V. EXPERIMENTAL RESULTS

In this section we present the experimental planning and the energy profile results. Firstly, the average values for time and energy consumption were obtained by performing 100 tests for each cryptographic primitive. The main goal was to be able to compare the energy consumption and execution times of each cryptographic primitive for different providers, various key lengths and several security modes. The tables with these values are available in Appendix. Secondly, to discover if the differences in the mean values are significant, a non-parametric Wilcoxon rank sum test [37] has been applied. This test is used to analyze two population means through the use of statistical examination. There are different tests depending on the data characteristics. For comparing two groups of related values that do not follow a normal distribution the non-parametric Wilcoxon rank sum test is used. We have used the statistical package R [38]. A difference is considered as significant if the significance level of the test is lower than $\alpha = 0.05$. The tables with the interpretation of the result of the Wilcoxon rank sum test are shown and discussed in this section. These tables are complemented with graphs that show the energy consumption versus time execution tradeoff. Only a subset of the generated graphs are discussed in the paper because, in general, the mean values shown in Appendix and the interpretation of the Wilcoxon rank sum test shown in this section provide information that can be interpreted more accurately than the visual observation of each individual test. Finally, tables including the p-values for the mean Wilcoxon test are not included in the paper to preserve readability. Both the p-values and the tradeoff graphs can be consulted at the following url: http://www.lcc.uma.es/~monte /CryptoConsumption /dataSet /p-ValuesandPlots.pdf. The measurement units used in our experiments throughout

TABLE 6. Symmetric key generation - RQ1: providers.

AES Key Generation						
	Consumption (mW)			Time (milliseconds)		
Key	BC-IAIK	BC-SC	IAIK-SC	BC-IAIK	BC-SC	IAIK-SC
128	⊕	⊙	⊖	⊖	⊙	⊕
192	⊕	⊙	⊖	⊖	⊖	⊙
256	⊕	⊙	⊖	⊖	⊖	⊕
DESede Key Generation						
	Consumption (mW)			Time (milliseconds)		
Key	BC-IAIK	BC-SC	IAIK-SC	BC-IAIK	BC-SC	IAIK-SC
112	⊙	⊙	⊙	⊙	⊙	⊕
168	⊙	⊙	⊙	⊙	⊙	⊕

TABLE 7. Asymmetric key generation - RQ1: providers.

RSA Key Generation						
	Consumption (mW)			Time (milliseconds)		
Key	BC-IAIK	BC-SC	IAIK-SC	BC-IAIK	BC-SC	IAIK-SC
512	⊕	⊙	⊖	⊕	⊕	⊖
1024	⊕	⊙	⊖	⊕	⊙	⊖
2048	⊕	⊙	⊖	⊕	⊙	⊖
DSA Key Generation						
	Consumption (mW)			Time (milliseconds)		
Key	BC-IAIK	BC-SC	IAIK-SC	BC-IAIK	BC-SC	IAIK-SC
1024	⊖	⊙	⊕	⊖	⊙	⊕
2048	NA	NA	⊕	NA	NA	⊕

TABLE 8. AES encrypt - RQ1: providers.

AES Encrypt						
ECB_PKCS5Padding (Default mode)						
	Consumption (mW)			Time (milliseconds)		
Key	BC-IAIK	BC-SC	IAIK-SC	BC-IAIK	BC-SC	IAIK-SC
128	⊙	⊙	⊙	⊙	⊙	⊙
192	⊙	⊙	⊙	⊙	⊖	⊙
256	⊙	⊙	⊙	⊙	⊙	⊙
CBC_PKCS5Padding (Recommended mode)						
	Consumption (mW)			Time (milliseconds)		
128	⊙	⊙	⊙	⊙	⊙	⊙
192	⊙	⊙	⊙	⊙	⊖	⊖
256	⊙	⊙	⊙	⊙	⊖	⊙

TABLE 9. AES decrypt - RQ1: providers.

AES Decrypt						
ECB_PKCS5Padding (Default mode)						
	Consumption (mW)			Time (milliseconds)		
Key	BC-IAIK	BC-SC	IAIK-SC	BC-IAIK	BC-SC	IAIK-SC
128	⊙	⊙	⊙	⊙	⊙	⊙
192	⊖	⊙	⊕	⊙	⊙	⊙
256	⊖	⊙	⊕	⊕	⊙	⊙
CBC_PKCS5Padding (Recommended mode)						
	Consumption (mW)			Time (milliseconds)		
128	⊙	⊙	⊙	⊙	⊙	⊙
192	⊖	⊙	⊕	⊙	⊙	⊙
256	⊙	⊙	⊙	⊙	⊙	⊙

TABLE 10. DESede encrypt - RQ1: providers.

DESede Encrypt						
ECB_PKCS5Padding (Default mode)						
	Consumption (mW)			Time (milliseconds)		
Key	BC-IAIK	BC-SC	IAIK-SC	BC-IAIK	BC-SC	IAIK-SC
112	⊙	⊙	⊙	⊙	⊙	⊙
168	⊖	⊙	⊕	⊙	⊙	⊙
CBC_PKCS5Padding (Recommended mode)						
	Consumption (mW)			Time (milliseconds)		
112	⊙	⊙	⊙	⊙	⊙	⊕
168	⊙	⊙	⊕	⊖	⊙	⊙

TABLE 11. DESede decrypt - RQ1: providers.

DESede Decrypt						
ECB_PKCS5Padding (Default mode)						
	Consumption (mW)			Time (milliseconds)		
Key	BC-IAIK	BC-SC	IAIK-SC	BC-IAIK	BC-SC	IAIK-SC
112	⊙	⊙	⊙	⊖	⊖	⊙
168	⊖	⊙	⊕	⊖	⊖	⊙
CBC_PKCS5Padding (Recommended mode)						
	Consumption (mW)			Time (milliseconds)		
112	⊙	⊙	⊙	⊖	⊖	⊙
168	⊙	⊙	⊙	⊖	⊖	⊖

the paper are milliwatts (mw) for energy and milliseconds (ms) for time.

A. OBJECTIVES AND RESEARCH QUESTIONS

The methodology of this study is defined according to the goal-question-metrics approach [39] as follows: *"Analyze cryptographic primitives offered by three different security providers for Android, from the point of view of software developers"*. To achieve this goal we set the following Research Questions (RQs):

RQ1. Which crypto provider is the most energy efficient for different operations of crypto primitives? This question aims to discover whether the power consumption of cryptographic primitives implemented by alternative providers is significantly different and whether one provider could be selected or not as the greenest one.

RQ2. What is the cost of increasing the security level in terms of power consumption? This question explores the influence of key length in the power consumption of alternative implementations of the cryptographic algorithms to identify the cases in which longer key lengths could be used without penalizing the power consumption.

RQ3. Which transformation, comparing the default and the recommended transformations, is the most energy efficient for different crypto providers in Android? As explained in Section IV, programmers who are non-experts in security will normally select the default transformation. However, it is not straightforward to know the default modes for each provider. RQ3 explores the relationships between security and power consumption by calculating the energy impact of replacing the default transformation with a more secure one. We answer all these questions for the cryptographic algorithms that are considered by security standards to be the most secure, as detailed in the following subsections. In the rest of the section, each subsection answers one of our research questions.

B. RQ1. WHICH CRYPTO PROVIDER IS THE MOST ENERGY EFFICIENT FOR DIFFERENT OPERATIONS OF CRYPTO PRIMITIVES?

Tables 6 – 18 show the interpretation of the results of applying the Wilcoxon rank sum test over the tables in Appendix. Concretely, they show the results of comparing the means of energy consumption and execution time for different providers. The columns in the tables indicate the comparison between the BC and the IAIK providers (BC-IAIK column), between the BC and the SC providers (BC-SC column)

TABLE 12. RSA encrypt - RQ1: providers.

RSA Encrypt						
ECB_PKCS1Padding (IAIK default mode and recommend)						
	Consumption (mW)			**Time (milliseconds)**		
Key	**BC-IAIK**	**BC-SC**	**IAIK-SC**	**BC-IAIK**	**BC-SC**	**IAIK-SC**
512	⊕	⊕	⊖	⊕	⊕	⊖
1024	⊕	⊙	⊖	⊕	⊙	⊖
2048	⊕	⊙	⊖	⊕	⊙	⊖
ECB_NoPadding (BC and SC default mode)						
	Consumption (mW)			**Time (milliseconds)**		
512	⊙	⊙	⊙	⊙	⊙	⊙
1024	⊙	⊙	⊙	⊙	⊕	⊕
2048	⊖	⊖	⊙	⊙	⊙	⊕
NONE_NoPadding (BC and SC default mode)						
	Consumption (mW)			**Time (milliseconds)**		
512	⊙	⊙	⊙	⊙	⊕	⊖
1024	⊙	⊙	⊙	⊙	⊕	⊖
2048	⊖	⊙	⊙	⊙	⊕	⊖

TABLE 13. RSA decrypt - RQ1: providers.

RSA Encrypt						
ECB_PKCS1Padding (IAIK default mode and recommend)						
	Consumption (mW)			**Time (milliseconds)**		
Key	**BC-IAIK**	**BC-SC**	**IAIK-SC**	**BC-IAIK**	**BC-SC**	**IAIK-SC**
512	⊙	⊖	⊖	⊙	⊙	⊙
1024	⊙	⊙	⊙	⊙	⊙	⊖
2048	⊙	⊙	⊕	⊕	⊕	⊖
ECB_NoPadding (BC and SC default mode)						
	Consumption (mW)			**Time (milliseconds)**		
512	⊙	⊙	⊙	⊖	⊙	⊙
1024	⊙	⊙	⊙	⊙	⊙	⊙
2048	⊙	⊙	⊙	⊙	⊙	⊙
NONE_NoPadding (BC and SC default mode)						
	Consumption (mW)			**Time (milliseconds)**		
512	⊖	⊖	⊙	⊖	⊙	⊙
1024	⊙	⊙	⊙	⊙	⊙	⊙
2048	⊙	⊙	⊙	⊙	⊙	⊙

TABLE 14. RSA sign - RQ1: providers.

RSA Sign						
NONE						
	Consumption (mW)			**Time (milliseconds)**		
Key	**BC-IAIK**	**BC-SC**	**IAIK-SC**	**BC-IAIK**	**BC-SC**	**IAIK-SC**
512	NA	NA	⊕	NA	NA	⊕
1024	NA	NA	⊙	NA	NA	⊙
2048	NA	NA	⊙	NA	NA	⊕
SHA-1						
	Consumption (mW)			**Time (milliseconds)**		
512	⊖	⊙	⊕	⊖	⊖	⊕
1024	⊙	⊙	⊙	⊖	⊖	⊕
2048	⊖	⊙	⊙	⊖	⊙	⊕

TABLE 15. RSA verify - RQ1: providers.

RSA Verify						
NONE						
	Consumption (mW)			**Time (milliseconds)**		
Key	**BC-IAIK**	**BC-SC**	**IAIK-SC**	**BC-IAIK**	**BC-SC**	**IAIK-SC**
512	NA	NA	⊙	NA	NA	⊙
1024	NA	NA	⊖	NA	NA	⊙
2048	NA	NA	⊙	NA	NA	⊙
SHA-1						
	Consumption (mW)			**Time (milliseconds)**		
512	⊙	⊙	⊙	⊖	⊙	⊕
1024	⊙	⊙	⊙	⊙	⊙	⊙
2048	⊙	⊙	⊙	⊖	⊖	⊙

and between the IAIK and the SC providers (IAIK-SC column), respectively. Thus, ⊕ indicates that the mean value when the second provider in the comparison is used is significantly greater than the mean value achieved when the first provider in the comparison is used, ⊖ signifies that the mean value is significantly less when the second provider is used and ⊙ signifies that there is no significant difference. The *NA* value is used to indicate that the comparison is 'Not Applicable'. The reference result that we wish to improve is the Android default cryptography library (BC).

In the rest of this section, the most interesting results for each cryptography primitive are discussed.

TABLE 16. DSA sign - RQ1: providers.

DSA Sign						
NONE						
	Consumption (mW)			**Time (milliseconds)**		
Key	**BC-IAIK**	**BC-SC**	**IAIK-SC**	**BC-IAIK**	**BC-SC**	**IAIK-SC**
1024	⊖	⊖	⊕	⊖	⊖	⊕
2048	NA	NA	⊙	NA	NA	⊕
SHA-1						
	Consumption (mW)			**Time (milliseconds)**		
1024	⊖	⊖	⊕	⊖	⊖	⊕
2048	NA	NA	⊙	NA	NA	⊙

TABLE 17. DSA verify - RQ1: providers.

DSA Verify						
NONE						
	Consumption (mW)			**Time (milliseconds)**		
Key	**BC-IAIK**	**BC-SC**	**IAIK-SC**	**BC-IAIK**	**BC-SC**	**IAIK-SC**
1024	⊙	⊙	⊙	⊖	⊙	⊙
2048	NA	NA	⊙	NA	NA	⊖
SHA-1						
	Consumption (mW)			**Time (milliseconds)**		
1024	⊙	⊙	⊙	⊖	⊙	⊙
2048	NA	NA	⊙	NA	NA	⊙

TABLE 18. MAC key generation - RQ1: providers.

MAC						
Key Generation						
	Consumption (mW)			**Time (milliseconds)**		
Key	**BC-IAIK**	**BC-SC**	**IAIK-SC**	**BC-IAIK**	**BC-SC**	**IAIK-SC**
MD5	⊖	⊕	⊕	⊖	⊙	⊕
SHA1	⊖	⊙	⊕	⊖	⊙	⊕
SHA224	NA	NA	⊕	NA	NA	⊕
SHA256	⊖	⊙	⊕	⊕	⊕	⊕
SHA384	⊖	⊙	⊕	⊖	⊖	⊕
SHA512	⊖	⊕	⊕	⊖	⊙	⊕
MAC						
	Consumption (mW)			**Time (milliseconds)**		
MD5	⊙	⊕	⊕	⊙	⊙	⊙
SHA1	⊕	⊙	⊖	⊕	⊙	⊖
SHA224	NA	NA	⊖	NA	NA	⊙
SHA256	⊙	⊙	⊙	⊕	⊕	⊙
SHA384	⊕	⊙	⊖	⊕	⊙	⊖
SHA512	⊕	⊙	⊖	⊙	⊖	⊖

1) CONFIDENTIALITY

The algorithms analyzed for confidentiality are AES, DESede and RSA. The operations are key generation, encrypt and decrypt. AES and DESede are symmetric algorithms, while RSA is asymmetric. The main results are discussed for each algorithm.

RQ1. AES algorithm: The values in Table 6 indicate that a tradeoff can be made between energy and time consumption for the key generation operation in AES, because IAIK is the provider that consumes most energy, but is also the provider

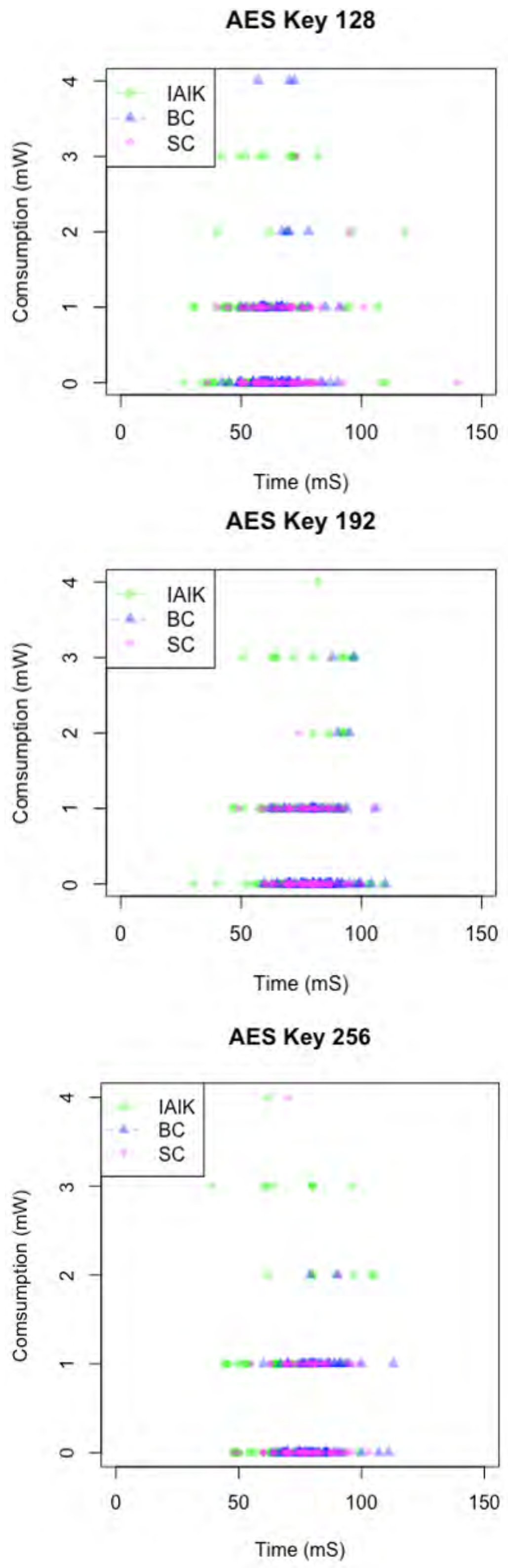

FIGURE 2. AES Key generation: Consumption vs. time tradeoff.

with the lowest execution time. The difference with other providers is significant in both cases. There are no important differences between the BC and the SC providers. This can also be observed in the graphs shown in Figure 2. For the encrypt operation (Table 8) the data indicates that there are no significant differences between the three providers, either in energy consumption or in time. However, for decryption (Table 9), IAIK is greener for some modes and key lengths, although the difference in the execution time is not significant. Concretely, in the ECB_PKCS5Padding mode (default mode) IAIK is greener for key lengths 192 and 256. In the CBC_PKCS5Padding (recommended mode), IAIK is greener for key length 192.

RQ1. DESede algorithm: Regarding energy consumption, Tables 6, 10, and 11 indicate that, in most cases, we cannot draw generic conclusions about which of the providers is the greenest one for the DESede algorithm. The reason is that the differences in energy consumption between the three providers are overall not significant for any operation. Only for encryption and decryption with a key size of 168 and in the default mode, does IAIK have better energy results. Comparing the execution times, the two external providers are better than BC in all the scenarios analyzed.

RQ1. RSA algorithm: As shown in Tables 7, 12, and 13, BC and SC are the best providers in most of the cases. IAIK consumes more energy and time for the key generation operation and the differences are not significant for the encrypt and decrypt operations. Only for the decrypt operation, with the default mode and a key size of 512 bits, SC is greener than BC. However, its execution time is greater.

In addition to the analysis of the tables with the interpretation of the results of the Wilcoxon rank test, for some cases it is also interesting to analyze some of the median values provided in the tables in Appendix. The most relevant results are discussed below.

RQ1. There are important differences in the energy profiles of the different algorithms for the key generation operation: The values for the key generation operation (Table 43 in Appendix) show that the energy profile of the RSA algorithm is completely different from the energy profiles of the AES and DESede algorithms. This is because the key generation for large primes performed by the RSA algorithm is time consuming and resource intensive, which has a high impact on the energy initially consumed by the application. For this reason, this is the most energy consuming algorithm for this operation.

RQ1. The IAIK provider is by far the provider which consumes the most for the key generation operation in RSA: Looking at the mean values in Table 43 we can observe that it is not only that the BC provider is the most energy efficient, with a consumption similar to the SC one. It is more important to note that in this case the IAIK provider almost quadruples the consumption of other suppliers in the recommended key length (2048).

RQ1. The IAIK provider is greenest for the decryption operation in AES: For the AES algorithm and the decryption operation, the greenest choice is the IAIK provider consuming 4 times less energy than others in the default mode (e.g., 0,15 mW instead of 0,60 mW, see Appendix). On the other hand, when using the DESede algorithm it is better to use an external provider, and not BC, because at least the execution time will be lower and sometimes the consumption of energy could also be lower.

We have seen how important it is to analyze the energy consumption and execution time separately for each crypto operation, and considering the frequency that the developer expects that one operation will be used. All these are useful hints for Android developers who now can take an energy-aware decision when choosing a provider, depending on the confidentiality operation that they expect will be most frequently used. For instance, if developers use the AES algorithm and use the decryption operation heavily, e.g., to access to a repository that stores encrypted data, then our study shows that the greenest choice is the IAIK provider, achieving energy savings four times lower than BC.

2) AUTHENTICATION

The algorithms analyzed for authentication are RSA and DSA. The operations are key generation, sign and verify. RSA and DSA are asymmetric algorithms. The results of the analysis for these algorithms are discussed below.

RQ1. RSA algorithm: The data in Tables 7, 13 and 13 indicates that the BC provider is better than IAIK for key generation, but the differences are not so significant for the sign and verify operations. Only in some modes is IAIK better for some specific key lengths for both energy consumption and time. For instance, for a key length of 512 bits IAIK consumes less energy and time in both the NONE and the SHA-1 modes. This can be easily observed in the graphs shown in Figure 3 for the sign operation and the SHA-1 mode. The previous point about the key generation using the IAIK provider and the RSA algorithm for confidentiality is also valid for the authentication primitive.

RQ1. DSA algorithm: When the DSA algorithm is applied, the IAIK provider is better in most of the cases. It is the greenest option for the key generation operation, both in energy consumption and time. It is also the best option for the sign operation when a key length of 1024 bits is used, independently of the mode. The results are not significant in the case of the verify operation. In this case, the graphs shown in Figure 4 cleary show that IAIK is the best option in all cases. In the first and second graphs all the measurements taken for the IAIK provider are localized in the lower left corner. This indicates that the energy and time consumption are much lower than for the BC and SC providers. The distribution of values for the IAIK provider are shown in the third graph using a different scale for both the time and the energy consumption.

Looking at the median values available in Table 43 in Appendix, there is an interesting result to discuss for the DSA algorithm and the IAIK provider.

RQ1. Key generation in the DSA algorithm: The analysis of the median values for the DSA algorithm indicates that for the key generation operation both the energy consumption and the time execution of the IAIK provider are extremely low in comparison to the other providers. For instance, for a key length of 1024 bits the energy consumption of the BC provider is 3296.22 mW, while with the IAIK provider it is 3.04 mW. This is a considerable difference that needs to be explained. Unfortunately, due to the IAIK license agreement we do not have access to the implementation. We figure that the IAIK provider pre-calculates and reuses some of the DSA parameters through different executions of the operation.

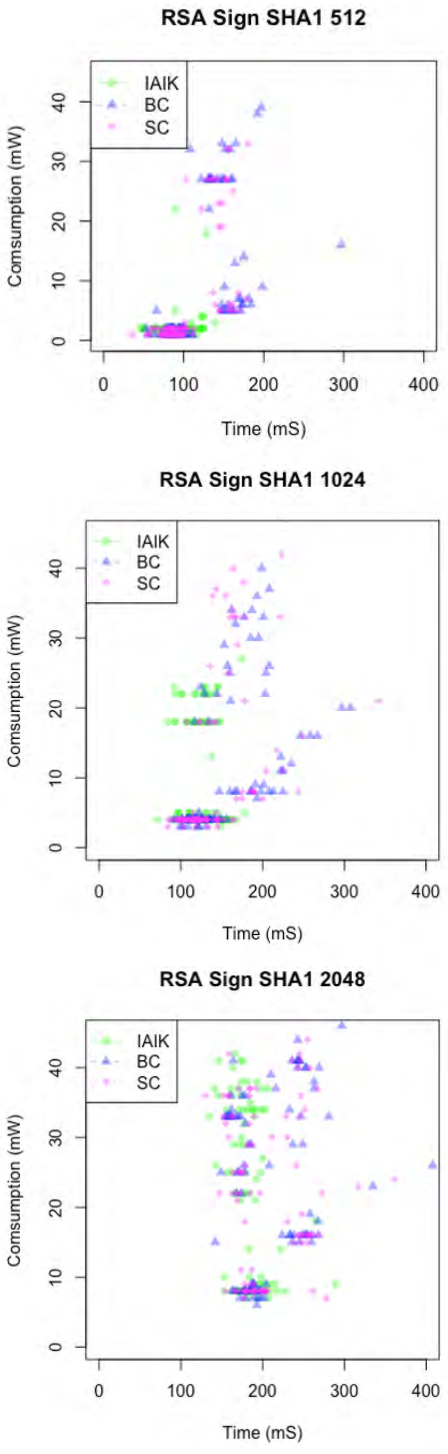

FIGURE 3. RSA Sign SHA1 mode: Consumption vs. time.

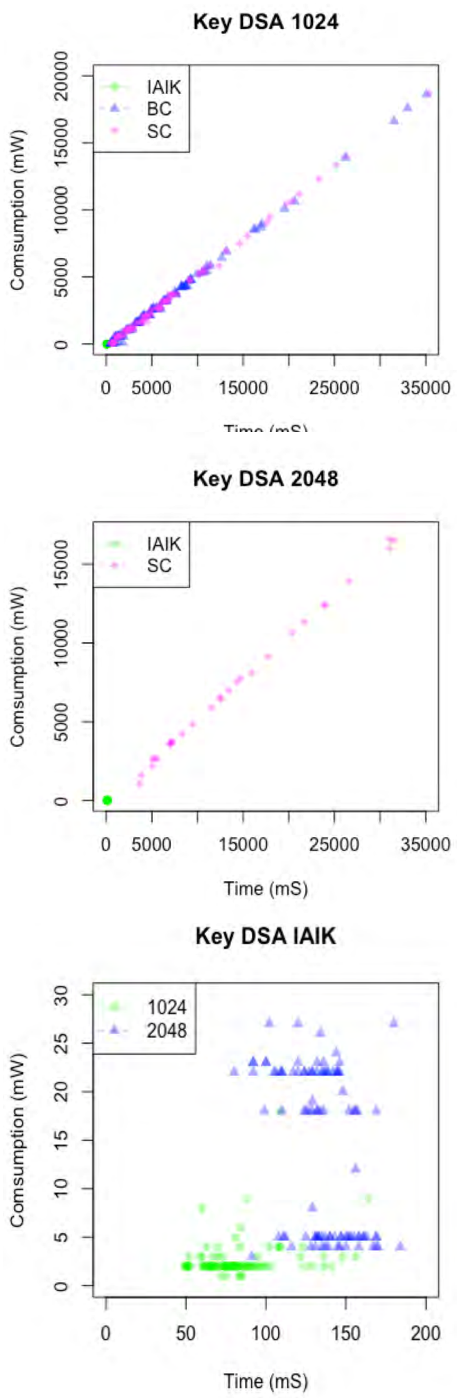

FIGURE 4. DSA key generation: Consumption vs. time tradeoff.

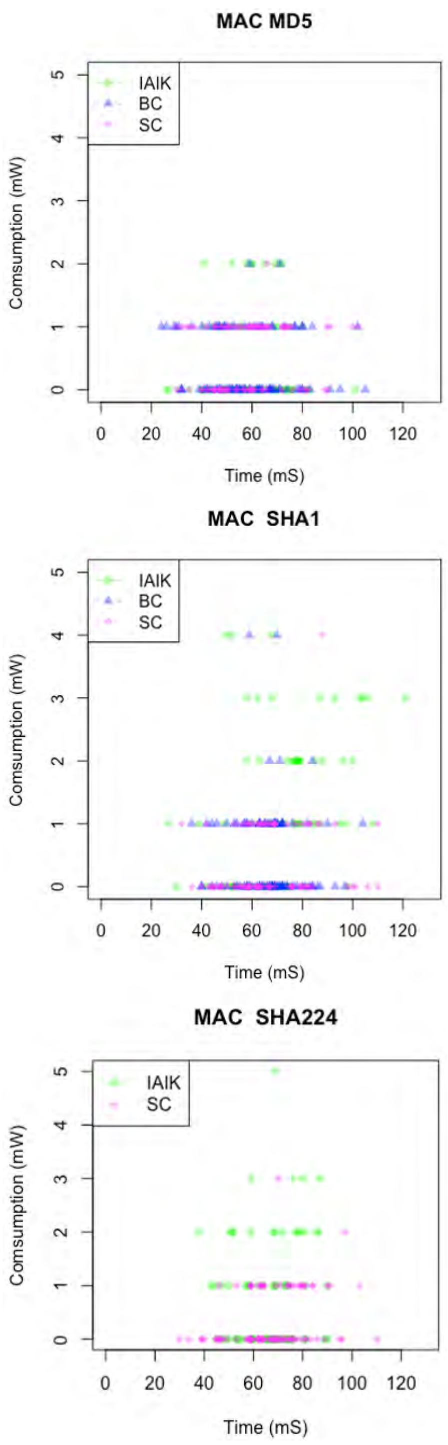

FIGURE 5. MAC operation: Consumption vs. time tradeoff.

This approach would reduce the execution time dramatically, and in consequence the energy expenditure would also be reduced.

3) INTEGRITY

Different algorithms with different digest lengths are analyzed for integrity. The operations are key generation and mac.

RQ1. MAC algorithm: As shown in Table 18, in this case IAIK is the greenest provider in all the scenarios analyzed for the key generation operation. However, the BC and the SC providers are the greenest in the case of the mac operation. The graphs in Figure 5 show the behavior of the three providers in the case of the mac operation. It is possible to observe that energy and time measurements are almost always higher for the IAIK provider.

TABLE 19. Symmetric key generation - RQ2: key length.

AES Key Generation						
	Consumption (mW)			Time (milliseconds)		
Key Length	**BC**	**IAIK**	**SC**	**BC**	**IAIK**	**SC**
128 - 192	⊙	⊙	⊙	⊕	⊕	⊕
192 - 256	⊙	⊙	⊙	⊙	⊖	⊙
128 - 256	⊙	⊙	⊙	⊕	⊕	⊕
DESede Key Generation						
	Consumption (mW)			**Time (milliseconds)**		
Key Length	**BC**	**IAIK**	**SC**	**BC**	**IAIK**	**SC**
112-168	⊙	⊙	⊙	⊕	⊕	⊕

TABLE 20. Asymmetric key generation - RQ2: key length.

RSA Key Generation						
	Consumption (mW)			Time (milliseconds)		
Key Length	**BC**	**IAIK**	**SC**	**BC**	**IAIK**	**SC**
512 - 1024	⊕	⊕	⊕	⊕	⊕	⊕
1024 - 2048	⊕	⊕	⊕	⊕	⊕	⊕
512 - 2048	⊕	⊕	⊕	⊕	⊕	⊕
DSA Key Generation						
	Consumption (mW)			**Time (milliseconds)**		
Key Length	**BC**	**IAIK**	**SC**	**BC**	**IAIK**	**SC**
1024 - 2048	NA	⊕	⊕	NA	⊕	⊕

TABLE 21. AES encrypt - RQ2: key length.

AES Encrypt						
ECB_PKCS5Padding (Default mode)						
	Consumption (mW)			**Time (milliseconds)**		
Key Length	**BC**	**IAIK**	**SC**	**BC**	**IAIK**	**SC**
128-192	⊕	⊙	⊙	⊙	⊙	⊙
192-256	⊙	⊙	⊙	⊙	⊙	⊙
128-256	⊕	⊙	⊙	⊙	⊙	⊙
CBC_PKCS5Padding (Recommended mode)						
	Consumption (mW)			**Time (milliseconds)**		
128-192	⊙	⊙	⊙	⊙	⊙	⊙
192-256	⊙	⊕	⊙	⊙	⊙	⊙
128-256	⊙	⊙	⊙	⊙	⊙	⊙

TABLE 22. AES decrypt - RQ2: key length.

AES Decrypt						
ECB_PKCS5Padding (Default mode)						
	Consumption (mW)			**Time (milliseconds)**		
Key Length	**BC**	**IAIK**	**SC**	**BC**	**IAIK**	**SC**
128-192	⊙	⊙	⊙	⊙	⊙	⊙
192-256	⊙	⊙	⊙	⊙	⊕	⊙
128-256	⊙	⊙	⊙	⊙	⊙	⊙
CBC_PKCS5Padding (Recommended mode)						
	Consumption (mW)			**Time (milliseconds)**		
128-192	⊙	⊙	⊙	⊙	⊙	⊙
192-256	⊕	⊕	⊙	⊙	⊙	⊙
128-256	⊕	⊕	⊙	⊙	⊙	⊙

TABLE 23. DESede encrypt - RQ2: key length.

DESede Encrypt						
ECB_PKCS5Padding (Default mode)						
	Consumption (mW)			**Time (milliseconds)**		
Key Length	**BC**	**IAIK**	**SC**	**BC**	**IAIK**	**SC**
112-168	⊙	⊙	⊙	⊙	⊙	⊙
CBC_PKCS5Padding (Recommended mode)						
	Consumption (mW)			**Time (milliseconds)**		
112-168	⊙	⊕	⊕	⊙	⊙	⊙

After this detailed analysis, the experimental data indicates that it is not possible to choose "the greenest provider". There are no big differences between BC and SC, which makes sense since BC is a variant of SC, slightly customized for Android. However, the data indicates that in many cases the IAIK external provider consumes less energy than the Android security library (BC), confirming our hypothesis that it is useful to explore different security providers. The recommendations that can be given for software developers regarding the security provider to be used in their applications are as follows:

Recommendations for RQ1: The recommendations are different for each cryptography primitive:

– Confidentiality: Our general recommendation is that IAIK is a good option from both the point of view of energy consumption and execution time. This provider only exhibits bad results for energy consumption for key generation, but this operation is less frequently used than encryption and decryption. The energy saving possibilities are the best for AES and sometimes for DESede, and for RSA; SC and IAIK are both good options. Note that the BC provider, included in Android releases is not the best option in either case.

– Authentication: Likewise for confidentiality, the IAIK provider is a good option in most cases, with the exception of the key generation operation for RSA, where the BC provider would be the best option. Developers can achieve the greatest energy savings with the DSA algorithm of the IAIK provider.

– Integrity: Contrary to previous cases, the IAIK is the greenest considering key generation, and not so good for the mac operation, but the differences are not very significant.

C. RQ2. WHAT IS THE COST OF INCREASING THE SECURITY LEVEL IN TERMS OF POWER CONSUMPTION?

In our experiments we increase the security level by increasing the key length. Tables 19–31 show the interpretation of the results of applying the Wilcoxon rank sum test over the Tables in Appendix, where the energy consumption and time execution for the same algorithm and different key lengths are compared. Concretely, each row indicates the results of comparing the two key lengths indicated in the first column of the tables. In this case, ⊕ indicates that the mean value for a key length is significantly greater than the mean value achieved when a lower key length is used, ⊖ signifies that the mean value is significantly less and ⊙ signifies that there are no significant differences. The *NA* value is used to indicate that the comparison is 'Not Applicable'.

The most interesting results for each crytographic primitive are discussed.

TABLE 24. DESede decrypt - RQ2: key length.

DESede Decrypt						
ECB_PKCS5Padding (Default mode)						
	Consumption (mW)			**Time (milliseconds)**		
Key Length	**BC**	**IAIK**	**SC**	**BC**	**IAIK**	**SC**
112-168	⊙	⊙	⊕	⊙	⊕	⊙
CBC_PKCS5Padding (Recommended mode)						
	Consumption (mW)			**Time (milliseconds)**		
112-168	⊙	⊙	⊙	⊙	⊙	⊙

TABLE 25. RSA encrypt - RQ2: key length.

RSA Encrypt						
ECB_PKCS1Padding (IAIK default mode and recommend)						
	Consumption (mW)			**Time (milliseconds)**		
Key Length	**BC**	**IAIK**	**SC**	**BC**	**IAIK**	**SC**
512-1024	⊕	⊕	⊕	⊕	⊕	⊙
1024-2048	⊕	⊕	⊕	⊕	⊕	⊕
512-2048	⊕	⊕	⊕	⊕	⊕	⊕
ECB_NoPadding (BC and SC default mode)						
	Consumption (mW)			**Time (milliseconds)**		
512-1024	⊙	⊕	⊕	⊙	⊙	⊕
1024-2048	⊕	⊕	⊕	⊕	⊕	⊕
512-2048	⊕	⊕	⊕	⊕	⊕	⊕
NONE_NoPadding (BC and SC default mode)						
	Consumption (mW)			**Time (milliseconds)**		
512-1024	⊕	⊕	⊕	⊕	⊕	⊕
1024-2048	⊕	⊕	⊕	⊙	⊙	⊕
512-2048	⊕	⊕	⊕	⊕	⊕	⊕

TABLE 26. RSA decrypt - RQ2: key length.

RSA Decrypt						
ECB_PKCS1Padding (IAIK default mode and recommend)						
	Consumption (mW)			**Time (milliseconds)**		
Key Length	**BC**	**IAIK**	**SC**	**BC**	**IAIK**	**SC**
512-1024	⊕	⊕	⊕	⊕	⊕	⊕
1024-2048	⊕	⊕	⊕	⊕	⊕	⊕
512-2048	⊕	⊕	⊕	⊕	⊕	⊕
ECB_NoPadding (BC and SC default mode)						
	Consumption (mW)			**Time (milliseconds)**		
512-1024	⊕	⊕	⊕	⊕	⊕	⊕
1024-2048	⊕	⊕	⊕	⊕	⊕	⊕
512-2048	⊕	⊕	⊕	⊕	⊕	⊕
NONE_NoPadding (BC and SC default mode)						
	Consumption (mW)			**Time (milliseconds)**		
512-1024	⊕	⊕	⊕	⊕	⊕	⊕
1024-2048	⊕	⊕	⊕	⊕	⊕	⊕
512-2048	⊕	⊕	⊕	⊕	⊕	⊕

TABLE 27. RSA sign - RQ2: key length.

RSA Sign						
NONE						
	Consumption (mW)			**Time (milliseconds)**		
Key Length	**BC**	**IAIK**	**SC**	**BC**	**IAIK**	**SC**
512-1024	NA	⊕	⊕	NA	⊕	⊕
1024-2048	NA	⊕	⊕	NA	⊕	⊕
512-2048	NA	⊕	⊕	NA	⊕	⊕
SHA-1						
	Consumption (mW)			**Time (milliseconds)**		
512-1024	⊕	⊕	⊕	⊕	⊕	⊕
1024-2048	⊕	⊕	⊕	⊕	⊕	⊕
512-2048	⊕	⊕	⊕	⊕	⊕	⊕

TABLE 28. RSA verify - RQ2: key length.

RSA Verify						
NONE						
	Consumption (mW)			**Time (milliseconds)**		
Key Length	**BC**	**IAIK**	**SC**	**BC**	**IAIK**	**SC**
512-1024	NA	⊕	⊙	NA	⊕	⊕
1024-2048	NA	⊕	⊕	NA	⊙	⊙
512-2048	NA	⊕	⊕	NA	⊕	⊕
SHA-1						
	Consumption (mW)			**Time (milliseconds)**		
512-1024	⊕	⊕	⊕	⊕	⊕	⊕
1024-2048	⊕	⊕	⊕	⊕	⊙	⊙
512-2048	⊕	⊕	⊕	⊕	⊕	⊕

TABLE 29. DSA sign - RQ2: key length.

DSA Sign						
NONE						
	Consumption (mW)			**Time (milliseconds)**		
Key Length	**BC**	**IAIK**	**SC**	**BC**	**IAIK**	**SC**
1024-2048	NA	⊕	⊕	NA	⊕	⊕
SHA-1						
	Consumption (mW)			**Time (milliseconds)**		
1024-2048	NA	⊕	⊕	NA	⊕	⊕

TABLE 30. DSA verify - RQ2: key length.

DSA Verify						
NONE						
	Consumption (mW)			**Time (milliseconds)**		
Key Length	**BC**	**IAIK**	**SC**	**BC**	**IAIK**	**SC**
1024-2048	NA	⊕	⊕	NA	⊕	⊕
SHA-1						
	Consumption (mW)			**Time (milliseconds)**		
1024-2048	NA	⊕	⊕	NA	⊕	⊕

1) CONFIDENTIALITY

As for RQ1, the algorithms analyzed for confidentiality are AES, DESede and RSA and the operations are key generation, encrypt and decrypt. In this case, the results highlight the important differences that exist between symmetric and asymmetric algorithms.

RQ2. Symmetric key generation (AES and DESede): The values in Table 19 indicates that energy consumption does not significantly increase when a longer key is used. The tendency is however different for execution time, where longer key lengths normally imply longer execution times. The only exception is from 192 to 256 key lengths, where the difference is not so significant. This can be observed in the graphs shown in Figure 2, especially for energy consumption, where the distribution of the measured values for the three providers are almost the same in the three graphs, i.e., independently of the key length.

RQ2. Symmetric encrypt/decrypt (AES and DESede): The values in Tables 21, 22, 23 and 24 indicate that, in general, execution time does not significantly increase when

TABLE 31. MAC - RQ2: key length.

MAC						
Key Generation						
	Consumption (mW)			**Time (milliseconds)**		
Key Length	**BC**	**IAIK**	**SC**	**BC**	**IAIK**	**SC**
MD5-SHA224	NA	⊙	⊙	NA	⊕	⊕
MD5-SHA256	⊕	⊙	⊙	⊙	⊕	⊕
MD5-SHA384	⊙	⊙	⊙	⊕	⊕	⊕
MD5-SHA512	⊙	⊙	⊙	⊕	⊕	⊕
SHA1-SHA224	NA	⊙	⊙	NA	⊕	⊖
SHA1-SHA256	⊙	⊙	⊙	⊖	⊕	⊖
SHA1-SHA384	⊙	⊕	⊙	⊙	⊕	⊖
SHA1-SHA512	⊙	⊙	⊙	⊙	⊕	⊙
MAC						
	Consumption (mW)			**Time (milliseconds)**		
MD5-SHA224	NA	⊕	⊖	NA	⊕	⊕
MD5-SHA256	⊕	⊕	⊙	⊙	⊕	⊕
MD5-SHA384	⊕	⊕	⊙	⊕	⊕	⊕
MD5-SHA512	⊕	⊕	⊙	⊕	⊕	⊕
SHA1-SHA224	NA	⊖	⊙	NA	⊖	⊙
SHA1-SHA256	⊙	⊖	⊕	⊖	⊙	⊙
SHA1-SHA384	⊕	⊙	⊕	⊙	⊙	⊙
SHA1-SHA512	⊕	⊕	⊕	⊕	⊙	⊙

a longer key is used for either the encrypt or the decrypt operations. This tendency is sometimes different for energy consumption, where encrypting and decrypting with longer key lengths normally implies more power consumption.

There are however a few exceptions to this conclusion that are worth mentioning: (1) Using the AES algorithm and the BC provider to encrypt, the energy consumption increases when changing from 128 to a longer length; the difference however is not significant when 256 is used instead of 192; (2) Using the DESede algorithm and the BC provider the longest key (168) can be used without energy penalization; however this is not the case when one of the other two providers is used; (3) Using the AES algorithm and the default mode to decrypt, the longest key length can be used at no cost from an energy point of view. However, if the recommended mode is used both the BC and the IAIK provider consumes more energy when the key length is increased, and (4) Using the DESede algorithm in the default mode to decrypt, the SC provider consumes more energy when the key length is longer and IAIK consumes more time.

RQ2. Asymmetric algorithms (RSA): For asymmetric algorithms the results are similar for the key generation, encrypt and decrypt operations. In all cases, both the energy consumption and the execution time increase when the key length increases (Table 20 and graphs in Figure 6).

It seems logical that the execution time, and therefore the energy consumption, increase when the key length is incremented. Morever, the Wilcoxon rank sum test indicates that these increments are significant. Thus, in this case it would be useful to analyze how large the increment is by looking at the mean values shown in the Tables in Appendix. From these values, some interesting results are obtained.

RQ2. Symmetric algorithms: The cost of increasing the key length and therefore the security is initially affordable

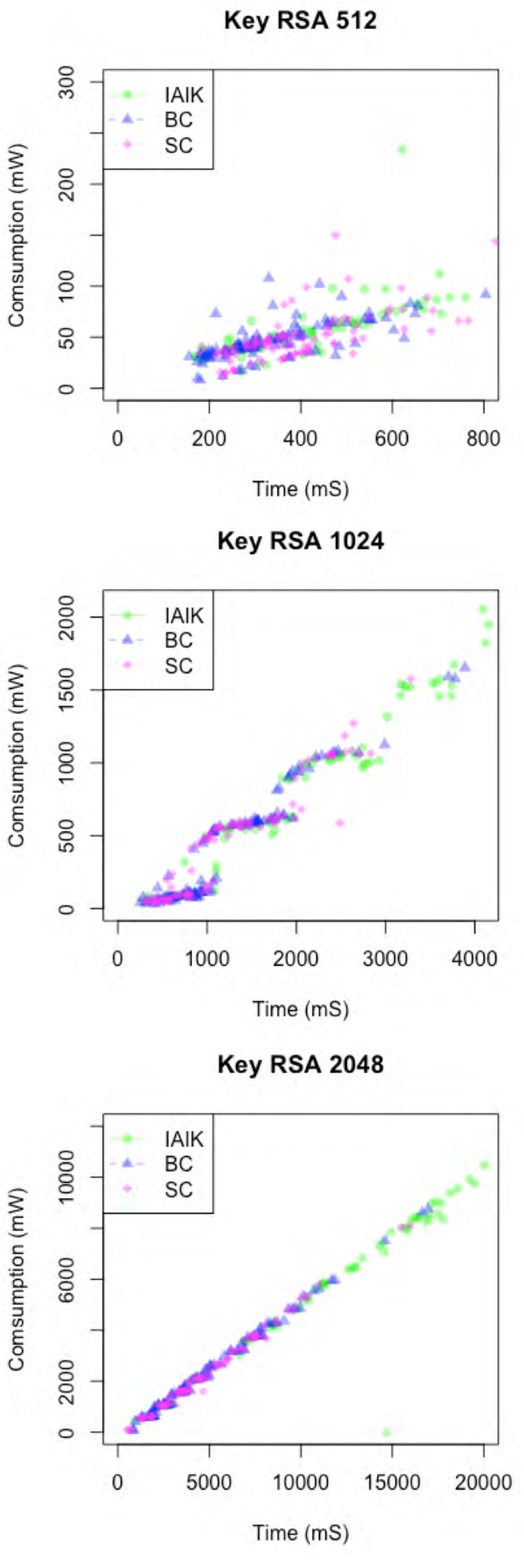

FIGURE 6. RSA Key generation: Consumption vs. time tradeoff.

both in terms of energy consumption and time execution. For instance, in Table 42 the differences between the mean values are less than 0.4 mW and 0.3 ms for the three providers.

RQ2. Asymmetric algorithms: In the case of the RSA algorithm there are huge differences between the means for short key lengths and the longest one. Thus, increasing the security in this case can be critical from the point of view of the energy consumption and the execution time. For instance, in Table 43 we can observe that for the IAIK provider there is a difference of more than 7000 mW and a difference of more than 15000 ms when the key length is increased from 512 to 2048. While also large, the difference is much smaller when the increment is from 512 to 1024. This means that the cost of increasing the key length from 512 to 1024 is affordable, but it is too costly to use a key length of 2028. This can be also observed in the graphs shown in Figure 6, especially when using a 2048 key length and the IAIK provider. In this case there is a clear correlation between the time required to generate the key and the energy consumption. Note that the scales used to represent the time and the consumption are different for each graph. So, here developers should make a trade off between the cost and the security level they want to achieve.

2) AUTHENTICATION

As for RQ1, the algorithms analyzed for authentication are RSA and DSA, but now different key lengths are compared. The results of the analysis for these algorithms are discussed below.

RQ2. RSA and DSA algorithms: As stated for confidentiality, for asymmetric algorithms the results are similar for the key generation, sign and verify operations. In all cases, both the energy consumption and the execution time increase when the key length increases (Tables 20, 27, 28, 29 and 30). Figure 6 illustrates this tendency for the key generation operation and Figure 7 for the decrypt operation.

Looking at the Tables in Appendix we can analyze how significant the increments are.

RQ2. Energy and time increments for key generation: The numbers in Table 43 indicate that for both algorithms it is very costly to move to a longer key. For RSA, the increment for BC or SC providers is very similar (around 2000 mW). However, the increment for the IAIK provider is much higher (8500 mW). The increments in execution times are of more than 3000 ms in all cases. For DSA, in BC the 2048 key length is not supported. For the SC provider it is very costly to move from a 1024 to a 2018 key length, both in energy and time. However, it is much more affordable when the IAIK provider is used (around 10 mW and 50 ms). The reason for this difference between IAIK and the other providers was explained in the answers to RQ1.

RQ2. Energy and time increments for sign/verify: The numbers in Tables 50, 51, 52 and 53 indicate that the increments in energy consumption and execution time for the sign and verify operations are much lower than for the key generation operation. However, this difference can be significant in the context of applications that apply the operations several times and when this is multiplied by a large number of users running the application.

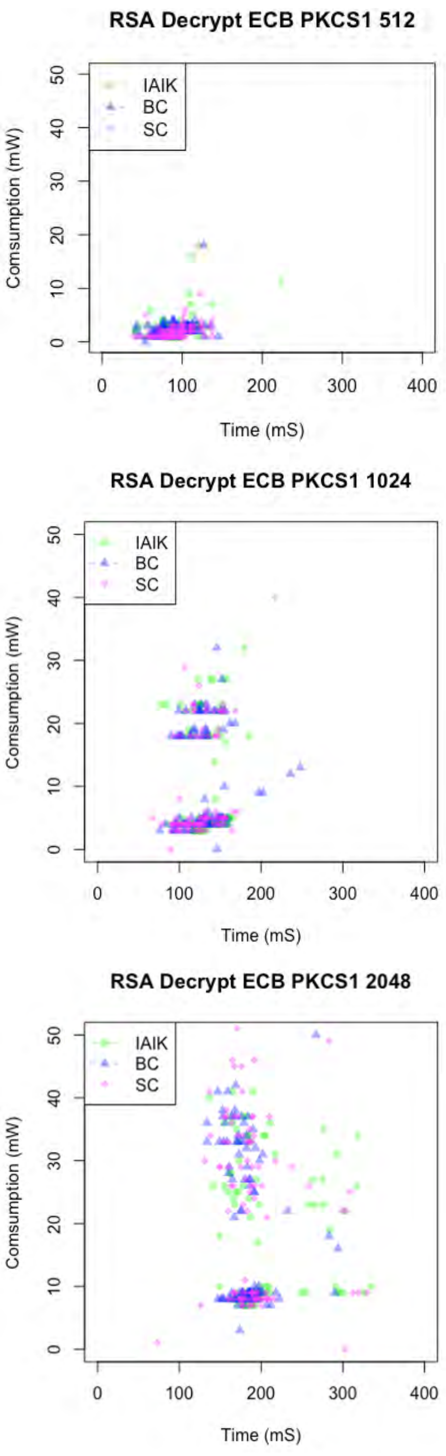

FIGURE 7. RSA Decrypt: Consumption vs. time tradeoff.

3) INTEGRITY

Different algorithms are analyzed for integrity, with different digest lengths (224, 256, 384 and 512 bits). The operations are key generation and mac. Currently, MD5 and SHA-1 are not recommended by security experts. In Table 31 we analyze the energy and time cost of moving from MD5 or SHA-1 to SHA2 algorithms with different digest length.

RQ2. Key generation operation: As shown in Table 31, for the key generation operation the use of algorithms that are more secure than MD5 or SHA-1 does not suppose a significant increment in energy consumption. There are only two exceptions when moving from MD5 to SHA-256 using BC and from SHA-1 to SHA-384 using IAIK. However, there is almost always an increment in the execution time. For instance, if the execution time is important the IAIK provider should not be selected, since the execution time increases in all the scenarios. Using the SC provider however, it is possible to move from SHA-1 to SHA-224, SHA-256 or SHA-384 without time penalization.

RQ2. Mac operation: The tendency is different for the mac operation. In this case, in an important number of cases there is a penalty in both energy consumption and execution time. There are however some cases for each provider where the energy and/or the time can be reduced.

Although both the energy consumption and the time increase when an algorithm with a larger digest length is used, in this case checking the mean values will help decide if the increment is justified.

RQ2. Increments in energy consumption and execution time when a larger digest length is used: In spite of the increments in the levels of both energy and time consumption, the values shown in Table 54 are not too large and, therefore, it is worth increasing the security by using a larger digest length. Although this may be different for each application, in general it seems that the small penalty on levels of energy and time consumption can be assumed.

Following this analysis we provide some key recommendations regarding the increase in security, using longer key lengths.

Recommendations for RQ2: The recommendations are different for confidentiality, authentication and integrity:

– Confidentiality: The recommendation is different depending on whether the algorithm is symmetric or asymmetric. For symmetric algorithms, programmers can increase the security of their applications using longer keys without huge variations in levels of energy or time consumption. However, for asymmetric algorithms developers must be aware of the security required by their applications since an increase in the key length involves a huge variation in the levels of energy and time consumption.

– Authentication: Developers can choose between BC or SC for RSA key generation, knowing that the cost of increasing security will not be so crucial. Otherwise, if the developer needs to create a DSA key, the IAIK provider shows the lowest consumption values. Keys generated with one provider can be used with the rest of the providers.

– Integrity: The developer can choose between recommended algorithms based on SHA-2, since the cost of incrementing security in energy is affordable and in time is almost null for some providers, e.g., SC. So, the choice can be made principally based on functional or cryptographic requirements rather than on energy expenditure.

TABLE 32. AES encrypt - RQ3: default mode.

AES Encrypt						
ECB_PKCS5Padding - CBC_PKCS5Padding						
	Consumption (mW)			Time (milliseconds)		
Key Length	BC	IAIK	SC	BC	IAIK	SC
128	⊕	⊙	⊙	⊙	⊙	⊙
192	⊙	⊕	⊙	⊙	⊙	⊙
256	⊙	⊙	⊙	⊙	⊙	⊙

TABLE 33. AES decrypt - RQ3: default mode.

AES Decrypt						
ECB_PKCS5Padding - CBC_PKCS5Padding						
	Consumption (mW)			Time (milliseconds)		
Key Length	BC	IAIK	SC	BC	IAIK	SC
128	⊙	⊙	⊙	⊙	⊙	⊙
192	⊙	⊙	⊙	⊙	⊙	⊙
256	⊙	⊕	⊙	⊙	⊖	⊙

TABLE 34. DESede encrypt - RQ3: default mode.

DESede Encrypt						
ECB_PKCS5Padding - CBC PKCS5Padding						
	Consumption (mW)			Time (milliseconds)		
Key Length	BC	IAIK	SC	BC	IAIK	SC
112	⊙	⊙	⊙	⊙	⊙	⊙
168	⊙	⊕	⊙	⊙	⊙	⊙

TABLE 35. DESede decrypt - RQ3: default mode.

DESede Decrypt						
ECB_PKCS5Padding - CBC_PKCS5Padding						
	Consumption (mW)			Time (milliseconds)		
Key Length	BC	IAIK	SC	BC	IAIK	SC
112	⊙	⊙	⊕	⊙	⊙	⊙
168	⊙	⊕	⊖	⊙	⊙	⊙

D. RQ3. WHICH TRANSFORMATION, COMPARING THE DEFAULT AND THE RECOMMENDED TRANSFORMATIONS, IS THE MOST ENERGY EFFICIENT FOR DIFFERENT CRYPTO PROVIDERS IN ANDROID?

Tables 32 – 41 show the results of applying the Wilcoxon rank sum test over the tables in Appendix, where the energy consumption and time execution for the same algorithm and different transformations are compared. Specifically, we compare the default transformation with those recommended by security experts. In this case, each row indicates the results of comparing the two transformations indicated in the second row at the top of the tables. For instance, the value 'ECB_PKCS5Padding - CBC_PKCS5Padding' in the second row of the Table 32 indicates that we are comparing the ECB_PKCS5Padding transformation, which is the default one, with the CBC_PKCS5Padding transformation, which is the recommended one. In this case, ⊕ indicates that the mean value for a recommended transformation is significantly greater than the mean value achieved when the default transformation is used, ⊖ signifies that the mean value is significantly less and ⊙ signifies that there are no

TABLE 36. RSA encrypt - RQ3: default mode.

RSA Encrypt						
ECB_NoPadding - ECB_PKCS1Padding						
	Consumption (mW)			Time (milliseconds)		
Key Length	BC	IAIK	SC	BC	IAIK	SC
512	$\oplus$	$\oplus$	$\oplus$	$\odot$	$\oplus$	$\oplus$
1024	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$
2048	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$
NONE_NoPadding - ECB_PKCS1Padding						
	Consumption (mW)			Time (milliseconds)		
512	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$
1024	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\odot$
2048	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$

TABLE 37. RSA decrypt - RQ3: default mode.

RSA Decrypt						
ECB_NoPadding - ECB_PKCS1Padding						
	Consumption (mW)			Time (milliseconds)		
Key Length	BC	IAIK	SC	BC	IAIK	SC
512	$\ominus$	$\odot$	$\ominus$	$\odot$	$\oplus$	$\odot$
1024	$\odot$	$\odot$	$\odot$	$\odot$	$\oplus$	$\odot$
2048	$\odot$	$\ominus$	$\oplus$	$\odot$	$\oplus$	$\odot$
NONE_NoPadding - ECB_PKCS1Padding						
	Consumption (mW)			Time (milliseconds)		
512	$\ominus$	$\odot$	$\odot$	$\oplus$	$\oplus$	$\oplus$
1024	$\odot$	$\odot$	$\ominus$	$\odot$	$\odot$	$\ominus$
2048	$\odot$	$\odot$	$\oplus$	$\odot$	$\oplus$	$\odot$

TABLE 38. RSA sign - RQ3: default mode

RSA Sign						
NONE - SHA-1						
	Consumption (mW)			Time (milliseconds)		
Key Length	BC	IAIK	SC	BC	IAIK	SC
512	NA	$\odot$	$\odot$	NA	$\oplus$	$\oplus$
1024	NA	$\odot$	$\odot$	NA	$\ominus$	$\odot$
2048	NA	$\odot$	$\odot$	NA	$\odot$	$\odot$

TABLE 39. RSA verify - RQ3: default mode.

RSA Verify						
NONE - SHA-1						
	Consumption (mW)			Time (milliseconds)		
Key Length	BC	IAIK	SC	BC	IAIK	SC
512	NA	$\odot$	$\oplus$	NA	$\oplus$	$\oplus$
1024	NA	$\oplus$	$\oplus$	NA	$\odot$	$\odot$
2048	NA	$\odot$	$\oplus$	NA	$\odot$	$\odot$

TABLE 40. DSA sign - RQ3: default mode.

DSA Sign						
NONE - SHA-1						
	Consumption (mW)			Time (milliseconds)		
Key Length	BC	IAIK	SC	BC	IAIK	SC
1024	$\odot$	$\odot$	$\odot$	$\oplus$	$\oplus$	$\oplus$
2048	NA	$\odot$	$\odot$	NA	$\oplus$	$\odot$

significant differences. The *NA* value is used to indicate that the comparison is 'Not Applicable'.

1) CONFIDENTIALITY

The AES, DESede and RSA algorithms are discussed. The only operations considered are encrypt and decrypt. For AES and DESede there is only one recommended transformation.

TABLE 41. DSA verify - RQ3: default mode.

DSA Verify						
NONE - SHA-1						
	Consumption (mW)			Time (milliseconds)		
Key Length	BC	IAIK	SC	BC	IAIK	SC
1024	$\odot$	$\odot$	$\odot$	$\oplus$	$\oplus$	$\oplus$
2048	NA	$\odot$	$\odot$	NA	$\odot$	$\odot$

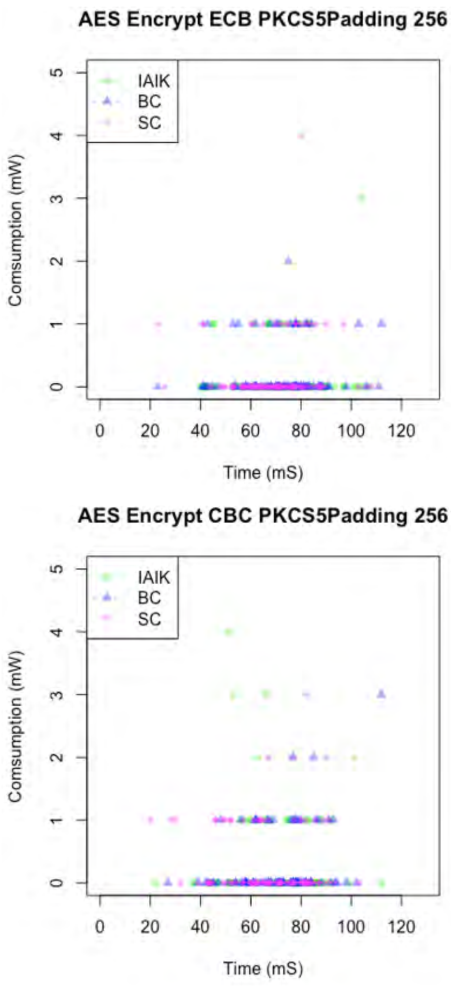

FIGURE 8. AES: Default vs. recommended transformation.

For the RSA two different recommended transformations are compared with the default one.

RQ3: AES algorithm: There are no significant differences in energy or time between the default mode and the recommended one when the encrypt or the decrypt operations are executed. With the SC provider a security increment can be done at no cost. For the other providers there are a few exceptions where a longer key implies an energy cost, but this cost is not very significant if we look at the mean values. So, the main conclusion here is that it is possible to follow the recommended transformations at a lower or even no cost. This is illustrated by the graphs in Figure 8, where the default and the recommended configurations are compared for a key length of 256. The graphs for other key lengths are similar.

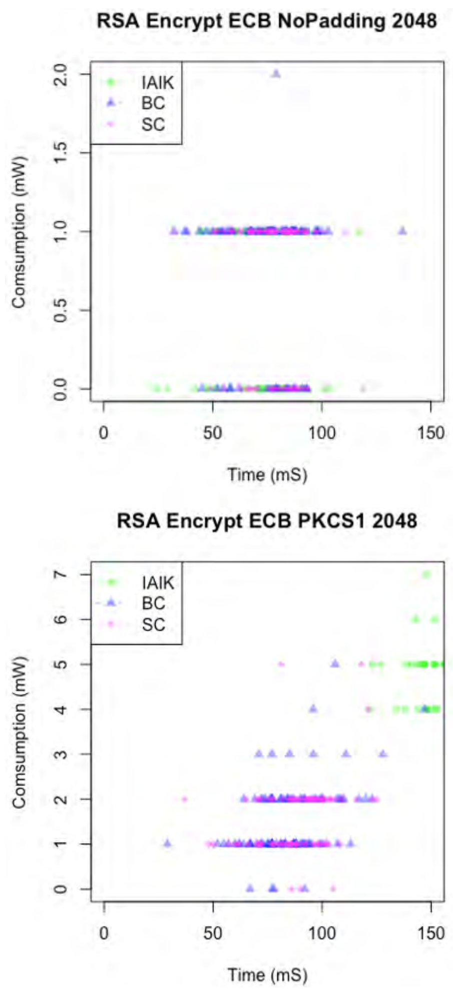

FIGURE 9. **RSA: Default vs. recommended transformation.**

RQ3: DESede algorithm: For the encrypt operation, a more secure mode can always be used without incurring a significant increase in energy or time; only using the IAIK provider and a key length of 168 the energy increases. For the decrypt operation, the differences in execution time are not significant for any of the scenarios analyzed. However, for energy consumption Table 35 should be checked because there are different tendencies depending on the chosen provider and key length.

RQ3: RSA algorithm: For the RSA algorithm, the results are completely different compared with those of the symmetric algorithms AES and DESede. The energy consumption and execution time increase significantly in all the cases for the encrypt operation, as illustrated in the graphs in Figure 9 when the default transformation is changed to one of the recommended transformations for a 2048 key length. For the decrypt operation, in the case of the IAIK provider, an increment in key size implies an increment in execution time, but not in energy. For the other providers, a fine-grained analysis needs to be done (see Table 37), however, taking a look at the mean values, the differences are not very important.

2) AUTHENTICATION

The RSA and DSA algorithms are analyzed for authentication. The operations are sign and verify. The default mode NONE and the recommended mode SHA-1 are compared for both algorithms.

RQ3: Some transformations are not supported in the Android default provider (BC): Using the RSA algorithm it is not possible to analyze the cost of moving from NONE to SHA-1 because the NONE mode is not supported in BC. For the DSA algorithm BC does not support a key length of 2048.

RQ3: RSA algorithm (with IAIK or SC providers): The behavior is different for the sign and verify operations. For the sign operation, the recommended mode can be used without a significant increase in the energy consumption. If the execution time is also relevant, the decision will depend on the provider and the key length, so a detailed analysis of the information in Table 38 must be done. However, for the verify operation the SC provider always increases the energy consumption when a more secure mode is used. When using the IAIK provider it will depend on the key length, and for 512 and 2018 there is no significant increase.

RQ3: DSA algorithm: A trade off between energy and time needs to be made for the DSA algorithm when a 1024 key length is used since the energy consumption does not significantly increase but the time does. For a key length of 2028 the SC provider could be used for both the sign and the verify operations to improve the security without incurring a significant increase in the energy or time consumption.

3) INTEGRITY

RQ3 is not applicable to this profile, since these algorithms do not require transformation to perform the cryptographic operation.

After this detailed analysis the main key recommendations regarding RQ3 are provided below.

RQ3: Key Recommendations:

– Confidentiality: In many of the cases analyzed, moving from the default transformation to the recommended one has similar energy consumption and execution time. Thus, there is no a justifiable reason to choose the default transformation instead of the one recommended by security experts, just the lack of knowledge about how to use other transformations. There are some exceptions for the encryption operation when trying to increase security in the RSA algorithm. In this case the developer should be aware that a better security level will negatively impact both on time and energy. So, it is recommended that the developer makes a trade off between security and cost, considering each particular provider.

– Authentication: The recommendation is different depending on the algorithm. For the DSA algorithm the results show that there is no penalty in energy when incrementing security, but for a key length of 1024 there is a penalty in time, for both sign and verify operations. So, if the response time is not crucial for the application, developers should use the transformations recommended by security experts. If the RSA algorithm is used, then a more fine grained analysis should be carried out to make the correct decision, for a specific key length and provider.

VI. THREATS TO VALIDITY

In this section, we discuss internal validity, reliability and external validity of the study presented in this paper. The internal validity intends to explore whether or not the energy results are influenced by other factors. The reliability is related to how the experiment was conducted and if others can replicate it with the same results. And finally, the external validity analyzes if the data obtained in the experiments can be generalized or not.

Regarding the internal validity, we should first analyze how precise the results obtained are. As discussed in Section III, we have opted for PowerTutor a software solution, instead of taking the measurements with tools implemented in the hardware. We chose PowerTutor because the results provided by this tool have been validated in a number of studies. Although hardware solutions provide more precise measurements, it is more difficult to be sure of which part of the software in execution is responsible for the consumption. To mitigate this we investigated the code in those cases when the results gave us an unexpected value, like for the key generation in the DSA algorithm for IAIK, which we have discussed in the previous section. Moreover, we were not interested in reporting absolute energy values, but rather to provide recommendations to developers based on comparative results.

Another threat to the internal validity of our experiment is related to analyzing whether the differences in energy consumption of the cryptographic algorithms are caused by the concrete implementations of the different security providers and not by other factors. To mitigate this threat we have used the same procedure and configuration parameters to measure energy data for all the alternative implementations.

Another internal threat is that the set of parameters we have considered in our experiment to answer the questions is not exhaustive, so there could be other parameters that influence the energy consumption of cryptographic algorithms implementations. Indeed, we have simplified this study for sign and cypher operations by using a fixed data length. We could have tested how the energy consumption changes when the data length varies. However, including a new parameter in our experiments would have severely complicated the comparative analysis and the answers to the proposed questions. We think that this simplification is not a threat to the validity of the results, but developers must be aware that the conclusions drawn in this paper can be considered valid only for a data length of 100 bytes. Exploring the influence of this parameter in the key results is part of our future work.

To mitigate reliability threats, different researchers have performed the data collection and set the analysis procedure. In addition, all the scripts and source code are available, and therefore anyone can reproduce our experiments and test the validity of our findings. Thanks to the use of a software energy measurement tool the reproducibility of our experiment is higher than others that use hardware solutions. We have carried out the tests over several months, and the results have always shown the same trends. Finally, other similar studies reproduce the experiments fewer than 30 times, but we opted to for 100 times, to provide more accurate results.

Concerning the external validity we have identified two threats. First, we have not tested our findings in third party applications. So, we do not consider the cost of invoking the cryptographic algorithms. We think that by not considering this, we can provide results which are independent of the application that is using these algorithms. However, we cannot report the energy savings of real applications that use our recommendations. This will be the next step in our research. The second threat is the generalization of the results to all mobile phones and Android versions. The problem here is that we need a reliable software energy measurement tool, able to measure nanoseconds, i.e., the execution time in smartphones with high performing CPUs (e.g., Snapdragon). As far as we know, this currently does not exist. To mitigate this we have used the most recent implementations of the external providers, which can be used with recent mobile phones and Android versions.

VII. CONCLUSIONS AND FUTURE WORK

The experiments carried out in this paper provided interesting information for software developers about how different cryptographics providers, algorithms and operations behave from an energy consumption point of view. According to the experimental data we can say that there is no crypto provider that can be considered the greenest one for all the algorithms, operations and transformations. Thus, the relevance of this study is in the fine-grained information that it provides and that can be used to make a reasoned decision about which is the best provider for the necessites of each application. It is also very useful to find that, for some providers, software developers can increase the level of security of their application without incurring too high an increment in the energy consumption. However, software developers need to be careful because this is not the case for all the providers. The analysis in this paper helps in making that decision. Finally, for some providers the default and the recommended transformations are the same, so non-expert software developers can use default transformations and be sure that their applications have an appropriate level of security. However, this is not true for all the providers so software developers interested in reducing energy consumpion need to be careful to avoid choosing those providers where an increment in security also implies a considerable increment in the energy consumption. Finally, another interesting conclusion is that the external providers consume less energy than the Android security library in many cases, confirming our hypothesis that it is useful to explore different security providers.

As part of our future work we plan to use these results to analyze if the security of current applications can be improved without incurring a huge penalty in either energy consumption or execution time. This will be done for real Android applications developed by us [41], [42] or by third parties where we will analyze the energy and time costs of changing between different providers, different key lengths and different transformation modes. The results shown by [16] already indicate that security can be considerably improved in current applications. Our goal is to complement that study to analyze whether the cost of increasing the level of security is affordable considering both energy consumption and execution time. The increments and reductions in both time execution and energy consumption shown in our study may seem insignificant when only one operation is measured, but even small values may have a huge impact on real applications, with large numbers of security operations and which are used by thousands of users.

APPENDIX
ENERGY CONSUMPTION AND EXECUTION TIME TABLES

In this appendix the average values for time and consumption are provided.

TABLE 42. Symmetric key generation- median.

AES Key Generation						
	Consumption (mW)			Time (milliseconds)		
Key Length	BC	IAIK	SC	BC	IAIK	SC
128	0.50	1.13	0.46	64.20	61.17	65.94
192	0.37	0.94	0.39	81.05	75.47	77.10
256	0.41	1.04	0.36	80.42	70.63	77.18
DESede Key Generation						
	Consumption (mW)			Time (milliseconds)		
Key Length	BC	IAIK	SC	BC	IAIK	SC
112	0.41	0.25	0.30	55.86	54.67	59.52
168	0.24	0.41	0.22	69.26	67.12	72.37

TABLE 43. Asymmetric key generation - median.

RSA Key Generation						
	Consumption (mW)			Time (milliseconds)		
Key Length	BC	IAIK	SC	BC	IAIK	SC
512	49.7	1136.5	51.8	349.3	2703.2	395.0
1024	415.8	1319.4	467.8	1228.7	3139.4	1276.7
2048	2416.5	9986.9	2325.5	5005.3	19744.4	4863.1
DSA Key Generation						
	Consumption (mW)			Time (milliseconds)		
Key Length	BC	IAIK	SC	BC	IAIK	SC
1024	3296.2	3.04	3142.3	6621.3	82.7	6359.1
2048	NA	13.9	45347.2	NA	134.9	85307.3

TABLE 44. AES encrypt - median.

AES Encrypt						
ECB_PKCS5Padding (Default mode)						
	Consumption (mW)			Time (milliseconds)		
Key Length	BC	IAIK	SC	BC	IAIK	SC
128	0.10	0.36	0.18	72.92	68.72	70.00
192	0.20	0.13	0.26	73.49	70.79	68.79
256	0.36	0.37	0.22	71.78	71.50	69.02
CBC_PKCS5Padding (Recommended mode)						
	Consumption (mW)			Time (milliseconds)		
128	0.54	0.33	0.29	69.91	68.15	66.86
192	0.58	0.24	0.27	72.34	71.57	65.83
256	0.43	0.33	0.34	71.20	70.21	68.52

TABLE 45. AES decrypt - median.

AES Decrypt						
ECB_PKCS5Padding (Default mode)						
	Consumption (mW)			Time (milliseconds)		
Key Length	BC	IAIK	SC	BC	IAIK	SC
128	0.48	0.26	0.41	69.73	69.78	69.16
192	0.60	0.15	0.25	66.41	67.72	67.67
256	0.70	0.18	0.38	65.89	70.92	68.84
CBC_PKCS5Padding (Recommended mode)						
	Consumption (mW)			Time (milliseconds)		
128	0.43	0.16	0.29	69.32	66.85	67.15
192	0.53	0.20	0.41	67.23	67.97	68.95
256	0.73	0.51	0.35	67.79	66.37	68.80

TABLE 46. DESede encrypt - median.

DESede Encrypt						
ECB_PKCS5Padding (Default mode)						
	Consumption (mW)			Time (milliseconds)		
Key Length	BC	IAIK	SC	BC	IAIK	SC
112	0.56	0.33	0.37	71.28	70.91	71.31
168	0.84	0.34	0.49	70.77	69.45	73.41
CBC_PKCS5Padding (Recommended mode)						
	Consumption (mW)			Time (milliseconds)		
112	0.41	0.31	0.45	69.71	67.56	72.73
168	0.49	0.47	0.90	71.14	67.15	70.34

TABLE 47. DESede decrypt - median.

DESede Decrypt						
ECB_PKCS5Padding (Default mode)						
	Consumption (mW)			Time (milliseconds)		
Key Length	BC	IAIK	SC	BC	IAIK	SC
112	0.54	0.56	0.37	76.22	66.41	65.34
168	0.67	0.42	0.57	77.73	69.94	67.26
CBC_PKCS5Padding (Recommended mode)						
	Consumption (mW)			Time (milliseconds)		
112	0.64	1.09	0.54	75.93	67.79	66.58
168	0.80	1.12	0.42	74.7	71.28	66.68

TABLE 48. RSA encrypt - median.

RSA Encrypt						
ECB_PKCS1Padding (IAIK default mode and recommend)						
	Consumption (mW)			Time (milliseconds)		
Key Length	BC	IAIK	SC	BC	IAIK	SC
512	0.53	1.22	0.82	71.46	82.76	77.15
1024	0.80	4.03	1.00	76.68	107.06	76.59
2048	1.55	18.77	1.97	85.97	176.69	87.56
ECB_NoPadding (BC and SC default mode)						
	Consumption (mW)			Time (milliseconds)		
512	0.23	0.19	0.21	71.29	69.04	71.36
1024	0.58	0.33	0.34	70.3	68.63	74.62
2048	0.96	0.66	0.76	76.35	73.99	79.20
NONE_NoPadding (BC and SC default mode)						
	Consumption (mW)			Time (milliseconds)		
512	0.17	0.24	0.19	58.19	60.74	62.25
1024	0.40	0.33	0.35	70.88	71.09	74.44
2048	0.91	0.63	0.73	68.59	70.52	79.08

TABLE 49. RSA decrypt - median.

RSA Decrypt						
ECB_PKCS1Padding (IAIK default mode and recommend)						
	Consumption (mW)			**Time (milliseconds)**		
Key Length	**BC**	**IAIK**	**SC**	**BC**	**IAIK**	**SC**
512	2.26	2.72	1.94	89.49	92.98	91.28
1024	10.75	9.42	9.11	130.26	132.98	125.63
2048	19.04	17.80	22.17	182.46	202.27	193.08
ECB_NoPadding (BC and SC default mode)						
	Consumption (mW)			**Time (milliseconds)**		
512	3.70	2.43	2.54	94.09	88.58	93.09
1024	9.79	9.81	10.47	130.39	128.03	125.74
2048	20.08	20.84	18.67	185.16	185.74	188.4
NONE_NoPadding				(BC and SC default mode)		
	Consumption (mW)			**Time (milliseconds)**		
512	4.01	2.41	2.31	85.26	79.93	83.58
1024	11.14	10.33	10.63	133.45	134.56	138.67
2048	20.27	20.32	18.50	186.79	190.97	185.63

TABLE 50. RSA sign - median.

RSA Sign						
NONE						
	Consumption (mW)			**Time (milliseconds)**		
Key Length	**BC**	**IAIK**	**SC**	**BC**	**IAIK**	**SC**
512	NA	3.08	5.94	NA	78.99	97.23
1024	NA	10.60	9.49	NA	132.83	141.03
2048	NA	19.78	22.00	NA	182.69	207.11
SHA-1						
	Consumption (mW)			**Time (milliseconds)**		
512	8.43	2.35	6.42	116.87	89.74	103.82
1024	11.74	9.83	10.75	155.31	125.41	141.50
2048	23.47	20.04	22.60	211.31	183.23	206.66

TABLE 51. RSA verify - median.

RSA Verify						
NONE						
	Consumption (mW)			**Time (milliseconds)**		
Key Length	**BC**	**IAIK**	**SC**	**BC**	**IAIK**	**SC**
512	NA	0.24	0.44	NA	54.33	56.31
1024	NA	0.66	0.41	NA	69.23	72.21
2048	NA	0.83	0.76	NA	73.44	75.24
SHA-1						
	Consumption (mW)			**Time (milliseconds)**		
512	0.48	0.68	0.47	70.94	67.32	70.57
1024	0.61	1.19	0.66	74.39	71.88	73.7
2048	1.00	1.36	1.05	80.61	75.46	76.23

TABLE 52. DSA sign - median.

DSA Sign						
NONE						
	Consumption (mW)			**Time (milliseconds)**		
Key Length	**BC**	**IAIK**	**SC**	**BC**	**IAIK**	**SC**
1024	7.85	2.82	5.08	112.31	88.75	104.19
2048	NA	13.23	12.79	NA	130.74	140.76
SHA-1						
	Consumption (mW)			**Time (milliseconds)**		
1024	7.91	3.01	4.96	121.79	98.15	111.1
2048	NA	13.38	13.95	NA	135.78	139.24

TABLE 53. DSA verify - median.

DSA Verify						
NONE						
	Consumption (mW)			**Time (milliseconds)**		
Key Length	**BC**	**IAIK**	**SC**	**BC**	**IAIK**	**SC**
1024	4.04	3.12	3.1	93.93	89.12	89.53
2048	NA	14.93	15.12	NA	151.9	147.01
SHA-1						
	Consumption (mW)			**Time (milliseconds)**		
1024	3.03	3.4	2.91	108.49	103.54	105.84
2048	NA	13.21	14.06	NA	151.41	150.24

TABLE 54. MAC - median.

MAC						
Key Generation						
	Consumption (mW)			**Time (milliseconds)**		
Key Length	**BC**	**IAIK**	**SC**	**BC**	**IAIK**	**SC**
MD5	0.77	0.47	1.13	65.51	57.81	66.52
SHA1	1.00	0.41	1.44	78.68	63.84	79.25
SHA224	NA	0.48	1.21	NA	66.91	74.50
SHA256	0.93	0.38	1.07	64.60	67.19	75.04
SHA384	1.50	0.56	1.15	78.77	68.37	73.21
SHA512	1.28	0.50	1.32	77.44	69.03	76.74
MAC						
	Consumption (mW)			**Time (milliseconds)**		
MD5	0.38	0.45	0.55	57.85	57.46	58.27
SHA1	0.5	1.05	0.37	66.35	70.78	66.67
SHA224	NA	0.64	0.34	NA	66.91	66.73
SHA256	0.55	0.66	0.74	55.56	68.53	67.39
SHA384	0.65	2.61	0.54	67.76	72.18	67.45
SHA512	0.64	2.49	0.54	70.78	72.93	66.36

REFERENCES

[1] C. Pang, A. Hindle, B. Adams, and A. E. Hassan, "What do programmers know about software energy consumption?" *IEEE Softw.*, vol. 33, no. 3, pp. 83–89, May/Jun. 2015, doi: 10.1109/MS.2015.83.

[2] G. Pinto, F. Castor, and Y. D. Liu, "Mining questions about software energy consumption," in *Proc. 11th Work. Conf. Mining Softw. Repositories (MSR)*, 2014, pp. 22–31.

[3] Q. Li and M. Zhou, "The survey and future evolution of green computing," in *Proc. IEEE/ACM Int. Conf. Green Comput. Commun. (GreenCom)*, Aug. 2011, pp. 230–233.

[4] J.-M. Horcas, M. Pinto, and L. Fuentes, "Green configurations of functional quality attributes," in *Proc. 21st Int. Syst. Softw. Product Line Conf. (SPLC)*, vol. A. Sevilla, Spain, Sep. 2017, pp. 79–83, doi: 10.1145/3106195.3106205.

[5] R. W. Ahmad, A. Gani, S. H. A. Hamid, F. Xia, and M. Shiraz, "A review on mobile application energy profiling: Taxonomy, state-of-the-art, and open research issues," *J. Netw. Comput. Appl.*, vol. 58, pp. 42–59, Dec. 2015, doi: http://dx.doi.org/10.1016/j.jnca.2015.09.002

[6] S. Hasan, Z. King, M. Hafiz, M. Sayagh, B. Adams, and A. Hindle, "Energy profiles of Java collections classes," in *Proc. 38th Int. Conf. Softw. Eng. (ICSE)*, 2016, pp. 225–236, doi: 10.1145/2884781.2884869.

[7] A. Banerjee, L. K. Chong, S. Chattopadhyay, and A. Roychoudhury, "Detecting energy bugs and hotspots in mobile apps," in *Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng. (FSE)*, 2014, pp. 588–598, doi: 10.1145/2635868.2635871.

[8] A. Castiglione, F. Palmieri, U. Fiore, A. Castiglione, and A. De Santis, "Modeling energy-efficient secure communications in multi-mode wireless mobile devices," *J. Comput. Syst. Sci.*, vol. 81, no. 8, pp. 1464–1478, 2015.

[9] A. Merlo, M. Migliardi, and L. Caviglione, "A survey on energy-aware security mechanisms," *Pervasive Mobile Comput.*, vol. 24, pp. 77–90, Dec. 2015, doi: 10.1016/j.pmcj.2015.05.005.

[10] D. S. A. Minaam, H. M. Abdual-Kader, and M. M. Hadhoud, "Evaluating the effects of symmetric cryptography algorithms on power consumption for different data types," *Int. J. Netw. Secur.*, vol. 11, no. 2, pp. 78–87, 2010, doi: 10.1007/s11367-011-0284-8.

[11] L. Zhang *et al.*, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proc. 8th IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synth. (CODES/ISSS)*, Oct. 2010, pp. 105–114, doi: 10.1145/1878961.1878982.

[12] A. Bakker, "Energy profiling android applications," in *Proc. 21st Twente Student Conf. IT*, Enschede, The Netherlands, Jun. 2014. [Online]. Available: http://fmt.cs.utwente.nl/files/sprojects/217.pdf

[13] H. Furusho, K. Hisazumi, T. Kamiyama, H. Inamura, T. Nakanishi, and A. Fukuda, "Power consumption profiling method based on android application usage," in *Information Science and Applications*. Heidelberg, Germany: Springer, 2015, pp. 891–898, doi: 10.1007/978-3-662-46578-3_106.

[14] K. Kim, D. Shin, Q. Xie, Y. Wang, M. Pedram, and N. Chang, ''FEPMA: Fine-grained event-driven power meter for android smartphones based on device driver layer event monitoring,'' in *Proc. Conf. Design, Autom. Test Eur. (DATE)*, 2014, pp. 367:1–367:6. [Online]. Available: http://dl.acm.org/citation.cfm?id=2616606.2617122

[15] A. Maiti, ''Flexible and effective energy management in smartphones,'' in *Proc. MobiSys PhD Forum (PhDForum)*, 2015, pp. 7–8, doi: 10.1145/2752746.2752792.

[16] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, ''An empirical study of cryptographic misuse in android applications,'' in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2013, pp. 73–84, doi: 10.1145/2508859.2516693.

[17] R. W. Ahmad *et al.*, ''A survey on energy estimation and power modeling schemes for smartphone applications,'' *Int. J. Commun. Syst.*, vol. 30, no. 11, p. e3234, 2017, doi: 10.1002/dac.3234.

[18] Y. Zhu and V. J. Reddi, ''GreenWeb: Language extensions for energy-efficient mobile Web computing,'' in *Proc. 37th ACM SIGPLAN Conf. Program. Lang. Design Implement. (PLDI)*, 2016, pp. 145–160, doi: 10.1145/2908080.2908082.

[19] X. Chen, A. Jindal, and Y. C. Hu, ''How much energy can we save from prefetching ads?: Energy drain analysis of top 100 apps,'' in *Proc. Workshop Power-Aware Comput. Syst., HotPower*, 2013, pp. 3:1–3:5, doi: 10.1145/2525526.2525848.

[20] M. Bokhari and M. Wagner, ''Optimising energy consumption heuristically on android mobile phones,'' in *Proc. Genetic Improvement Workshop*, Denver, CO, USA, 2016, pp. 1139–1140. [Online]. Available: http://cs.adelaide.edu.au/~markus/pub/2016-gecco-gi-energy.pdf, doi: 10.1145/2908961.2931691.

[21] N. R. Potlapally, S. Ravi, A. Raghunathan, and N. K. Jha, ''Analyzing the energy consumption of security protocols,'' in *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, Aug. 2003, pp. 30–35, doi: 10.1145/871506.871518.

[22] X. Liu and F. Qian, ''Measuring and optimizing android smartwatch energy consumption: Poster,'' in *Proc. 22nd Annu. Int. Conf. Mobile Comput. Netw. (MobiCom)*, 2016, pp. 421–423, doi: 10.1145/2973750.2985259.

[23] D. Li, S. Hao, J. Gui, and W. G. J. Halfond, ''An empirical study of the energy consumption of android applications,'' in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, 2014, pp. 121–130, doi: 10.1109/ICSME.2014.34.

[24] C. Bunse and A. Rohdé, ''Software development guidelines for performance and energy: Initial case studies,'' in *Advances and New Trends in Environmental Informatics*. Cham, Switzerland: Springer, 2017, pp. 25–35, 10.1007/978-3-319-44711-7_3.

[25] R. Jabbarvand, A. Sadeghi, H. Bagheri, and S. Malek, ''Energy-aware test-suite minimization for Android apps,'' in *Proc. 25th Int. Symp. Softw. Test. Anal. (ISSTA)*, 2016, pp. 425–436. [Online]. Available: http://delivery.acm.org/10.1145/2940000/2931067/p425-jabbarvand.pdf?ip=150.214.108.144&id=2931067&acc= CHORUS&key=DD1EC5BCF38B3699.9DD43FB5F2FDB30A.4D4702B0C3E38B35.6D218144511F3437&CFID=989679241&CFTOKEN= 43729866&__acm__=1506584542_2605a30dfdc37762469ba3553a284c46, doi: 10.1145/2931037.2931067.

[26] C. A. Lara-Niño, M. Morales-Sandoval, and A. Díaz-Pérez, ''An evaluation of AES and present ciphers for lightweight cryptography on smartphones,'' in *Proc. IEEE Int. Conf. Electron., Commun. Comput. (CONIELECOMP)*, Feb. 2016, pp. 87–93, doi: 10.1109/CONIELECOMP.2016.7438557.

[27] L. Caviglione, M. Gaggero, E. Cambiaso, and M. Aiello, ''Measuring the energy consumption of cyber security,'' *IEEE Commun. Mag.*, vol. 55, no. 7, pp. 58–63, Jul. 2017, doi: 10.1109/MCOM.2017.1600955.

[28] D. González, O. Esparza, J. L. Muñoz, J. Alins, and J. Mata, ''Evaluation of cryptographic capabilities for the Android platform,'' in *Future Network Systems and Security*. Cham, Switzerland: Springer, 2015, pp. 16–30, doi: 10.1007/978-3-319-19210-9_2.

[29] A. Ometov *et al.*, ''Feasibility characterization of cryptographic primitives for constrained (wearable) IoT devices,'' in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops (PerCom Workshops)*, Mar. 2016, pp. 1–6, doi: 10.1109/PERCOMW.2016.7457161.

[30] G. C. C. F. Pereira *et al.*, ''Performance evaluation of cryptographic algorithms over IoT platforms and operating systems,'' *Secur. Commun. Netw.*, vol. 2017, Aug. 2017, Art. no. 2046735, doi: 10.1155/2017/2046735.

[31] *When Mobile Apps Use Too Much Power. A Developer Guide for Android App Performance*, Qualcomm Technol. Inc., San Diego, CA, USA, Dec. 2013. [Online]. Available: https://developer.qualcomm.com/download/trepn-whitepaper-power.pdf

[32] J. R. Weiss, *Java Cryptography Extensions: Practical Guide for Programmers*. San Mateo, CA, USA: Morgan Kaufmann, 2004.

[33] *The Keyed-Hash Message Authentication Code (HMAC)*, document FIPSPUB198-1, 2008.

[34] *Advanced Encryption Standard (AES)*, document FIPS-197, 2001.

[35] *Digital Signature Standard (DSS)*, document FIPS-186-4, 2013.

[36] E. Barker and N. Mouha, ''Recommendation for the triple data encryption algorithm (TDEA) block cipher,'' National Institute of Standards and Technology, Gaithersburg, MD, USA, NIST Special Pub. 800-67, Revision 2, Nov. 2017, doi: 10.6028/NIST.SP.800-67r2.

[37] C. J. Wild and G. A. F. Seber, *Chance Encounters: A First Course in Data Analysis and Inference*. Hoboken, NJ, USA: Wiley, 1999.

[38] G. Grolemund and H. Wickham, *R for Data Science*. Newton, MA, USA: O'Reilly Media, 2016.

[39] V. R. Basili, G. Caldiera, and H. D. Rombach, ''The goal question metric approach,'' in *Encyclopedia of Software Engineering*. Hoboken, NJ, USA: Wiley, 1994.

[40] P. Rogaway, ''Evaluation of some blockcipher modes of operation,'' Dept. Comput. Sci., Univ. California, Davis, Davis, CA, USA, Tech. Rep., Feb. 2011. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.591.7222&rep=rep1&type=pdf

[41] I. Ayala, L. Mandow, M. Amor, and L. Fuentes, ''A mobile and interactive multiobjective urban tourist route planning system,'' *J. Ambient Intell. Smart Environ.*, vol. 9, no. 1, pp. 129–144, 2017, 10.3233/AIS-160413.

[42] I. Ayala, M. Amor, M. Pinto, L. Fuentes, and N. Gámez, ''iMuseumA: An agent-based context-aware intelligent museum system,'' *Sensors*, vol. 14, no. 11, pp. 21213–21246, 2014, doi: 10.3390/s141121213.

JOSÉ A. MONTENEGRO received the M.S. and Ph.D. degrees from the Department of Lenguajes y Ciencias de la Computación, Universidad de Málaga, Spain, in 2001 and 2006, respectively, and the M.B.A. degree from the Universidad Nacional de Educación a Distancia, Spain, in 2006. He was a Guest Researcher (Fulbright Scholar) with the National Institute of Standards and Technology, USA, from 2007 to 2009. He has been an Associate Professor with the Department of Lenguajes y Ciencias de la Computación, Universidad de Málaga, since 2010. His main research interests are in artificial intelligence and applied cryptography.

MÓNICA PINTO received the M.Sc. degree in computer science and the Ph.D. degree from the Universidad de Málaga, Spain, in 1998 and 2004, respectively. She is currently an Associate Professor with the Department of Lenguajes y Ciencias de la Computación, Universidad de Málaga. Her main research areas are energy-aware software development, component-based software engineering, aspect-oriented software development, architecture description languages, model-driven development, and context-aware mobile middlewares.

LIDIA FUENTES received the M.Sc. and Ph.D. degrees in computer science from the Universidad de Málaga, Spain, in 1992 and 1998, respectively. She has been with the Department of Lenguajes y Ciencias de la Computación, Universidad de Málaga, as a Lecturer and an Associate Professor since 1993 and a Full Professor since 2011, where she is currently the Head of the CAOSD Research Group. Her main research areas are energy-aware software development, aspect-oriented software development, model-driven development, software product lines, agent-oriented software engineering, self-adaptive middleware platforms, architecture description languages, and domain specific languages.