Implementation Issues in the Construction of Standard and Non-Standard Cryptography on Android Devices

Alexandre Melo Braga, Eduardo Moraes de Morais
Centro de Pesquisa e Desenvolvimento em Telecomunicações (Fundação CPqD)
Campinas, São Paulo, Brazil
{ambraga,emorais}@cpqd.com.br

Abstract—**This paper describes both the design decisions and implementation issues concerning the construction of a cryptographic library for Android Devices. Four aspects of the implementation were discussed in this paper: selection of cryptographic primitives, architecture of components, performance evaluation, and the implementation of non-standard cryptographic algorithms. The motivation behind the special attention given to the selection of alternative cryptographic algorithms was the recently revealed weakness found in international encryption standards, which may be intentionally included by foreign intelligence agencies.**

Keywords-Cryptography; Surveillance; Security; Android.

I. INTRODUCTION

Currently, the proliferation of smartphones and tablets and the advent of cloud computing are changing the way software is being developed and distributed. Additionally, the use in software systems of cryptographic techniques is increasing as well.

This paper discusses the construction of a cryptographic library for Android devices. The paper focuses on design decisions as well as on implementation issues of both standard and non-standard algorithms. This work contributes to the state of the practice by discussing the technical aspects and challenges of cryptographic implementations. This work is part of an effort to build security technologies into an integrated framework for mobile device security [2]. The evaluation of several cryptographic libraries on Android devices was reported in a previous work [1], showing that there is a lack of sophisticated cryptographic primitives, such as elliptic curves and bilinear pairings. Moreover, the majority of assessed schemes implements only standard algorithms, and, as far as authors know, there is no practical design that concerns alternative, non-standard cryptography.

The motivation behind the special attention given to the selection of alternative cryptographic algorithms was the recently revealed weakness, which may be intentionally included by foreign intelligence agencies in international encryption standards [16][26]. This fact alone raises doubt on all standardized algorithms, which are internationally adopted. In this context, a need arose to treat what has been called “alternative” or “non-standard” cryptography in opposition to standardized cryptographic schemes. The final intent was strengthening the implementation of advanced cryptography and fostering their use. Non-standard cryptography provides advanced mathematical concepts, such as bilinear pairings and elliptic curves, which are not fully standardized by foreign organizations, and suffer constant improvements.

The remaining parts of the text are organized as follows. Section II offers background on the subject of cryptographic implementation on Java and Android. Section III details the implementation aspects. Section IV presents a performance evaluation and comparison with other libraries. Section V concludes this text.

II. BACKGROUND AND RELATED WORK

This section briefly describes topics of interest: the Java Cryptographic Architecture (JCA) as a framework for pluggable cryptography; the Java Virtual Machine (JVM) with its Garbage Collector (GC) and Just-in-Time (JiT) compilation; and The Dalvik Virtual Machine (DVM) for Android.

A. *JCA*

The JVM is the runtime software ultimately responsible for the execution of Java programs. In order to be interpreted by JVM, Java programs are translated to bytecodes, an intermediary representation that is neither source code nor executable. The JCA [17] is a software framework for use and development of cryptographic primitives in the Java platform. The JCA defines, among other facilities, Application Program Interfaces (APIs) for digital signatures and secure hash functions [17]. On the other hand, APIs for encryption, key establishment and message authentication codes (MACs) are defined in the Java Cryptography Extension (JCE) [19]. Since version 1.4, the JCE was incorporated by JCA, being treated in practice as a single framework, named JCA or JCE [20].

The benefit of using a software framework, such as JCA, is to take advantage of good design decisions, reusing the whole architecture. The API keeps the same general behavior regardless of specific implementations. The addition of new algorithms is facilitated by the use of a standard API [20].

B. *GC on JVM*

An architectural feature of the JVM has great influence in the general performance of applications: the GC [35][37]. Applications have different requirements of GC. For some applications, pauses during garbage collection may be tolerable, or simply obscured by network latencies, in such a way that throughput is an important metric of performance.

Copyright (c) IARIA, 2014. ISBN: 978-1-61208-376-6

However, in others, even short pauses may negatively affect the user experience.

One of the most advertised advantages of JVM is that it shields the developer from the complexity of memory allocation and garbage collection. However, once garbage collection is a major bottleneck, it is worth understanding some aspects of its implementation.

The JVM incorporates a number of different GC algorithms that are combined using generational collection. While simple GC examines every live object in the heap, generational collection explores other hypothesis in order to minimize the work required to reclaim unused objects. The hypothesis supporting generation GC is corroborated by observed behavior of applications, where most objects survive for only a short period of time. Some objects can be reclaimed soon by memory management, because they have died shortly after being allocated. For example, iterator objects are often alive for the duration of a single loop.

On the other hand, some objects do live longer. For instance, there are typically some objects allocated at initialization that live until the program terminates. Between these two extremes are objects that live for the duration of some intermediate computation. For example, external loop variables live longer than inner loop variables. Efficient GC is made possible by focusing on the fact that a majority of objects die young.

Collections are clearly identified in diagrams, as shown in Figure 1. The figure shows the time consumed by the first 500 of 10.000 executions of pure-Java implementation of the AES encryption algorithm.

C. JiT Compilation

Other import consideration on performance of Java programs is the JiT Compilation [10][35]. Historically, Java bytecode used to be fully interpreted by the JVM and presented serious performance issues. Now a days, the technology known as HotSpot uses JiT Compilation not only to compile Java programs, but also to optimize them, while they execute. The result of JiTC is an application that has portions of its bytecode compiled and optimized for the targeted hardware, while other portions are still interpreted.

It is interesting to notice that JVM has to execute the code before to learn how to optimize it. The very first moments of an application show a relatively poor performance, since the bytecode is been interpreted, analyzed for optimizations, and compiled at the same time.

After this short period, the overall performance of the application improves and the execution tends to stabilize at an acceptable level of performance. Once again, the period of optimization and compilation is clearly identified in diagrams, as is shown in Figure 1.

A feature referred by Oracle as JVM Ergonomics was introduced in Java 5.0 with the goal of providing good performance with little or no tuning of configuration options for JVM. Instead of using fixed defaults, JVM ergonomics automatically selects GC, heap size, and runtime compiler at JVM startup. The result of ergonomics is that the choice of a GC does not matter to most applications. That is, most applications can perform well under the choices made by JVM, even in the presence of pauses of modest frequency and duration.

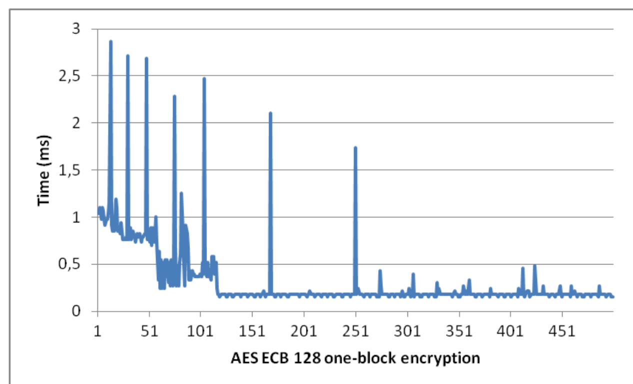

Figure 1. JiT Optimization of an AES execution.

Unfortunately, there is a potential negative side to security in the massive use of JiT Compilation. Security controls put in place into source code, in order to avoid side-channels, can be cut off by JiT optimizations. JiTC is not able to capture programmer's intent that is not explicitly expressed by Java's constructs. That is exactly the case of constant time computations needed to avoid timing attacks. Security-ware optimizations should be able to preserve security decisions and not undo protections, when transforming source code for cryptographic implementations to machine code. Hence, to achieve higher security against this kind of attacks, it is not recommended to use JiTC technology, what constitutes a trade-off between security and performance.

D. DVM

The DVM [7] is the virtual hardware that executes Java bytecode in Android. DVM is quite different from the traditional JVM, so that software developers have to be aware of those differences, and performance measurements over a platform independent implementation have to be taken in both environments.

Compared to JVM, DVM is a relatively young implementation and did not suffered extensive evaluation. In fact, the first independent evaluation of DVM was just recently published [13]. There are three major differences between DVM and JVM. First of all, DVM is a register-based machine, while JVM is stack-based. Second, DVM applies trace-based JiTC, while JVM uses method-based JiTC. Finally, former DVM implementations use mark-and-sweep GC, while current JVM uses generation GC.

Also, results from that DVM evaluation [13] suggest that current implementations of DVM are slower than current implementations of JVM. Concerning cryptographic requirements, a remarkable difference between these two environments is that the source of entropy in DVM is significantly different from the one found on JVM.

III. Description of the Implementation

In order to facilitate the portability of the cryptographic library for mobile devices, in particular for the Android

platform, the implementation was performed according to standard cryptographic API for Java, the JCA, its name conventions, and design principles [14][17]-[20].

Once JCA was defined as the architectural framework, the next design decision was to choose the algorithms minimally necessary to a workable cryptographic library. The current version of this implementation is illustrated by Figure 2 and presents the cryptographic algorithms and protocols described in the following paragraphs. The figure shows that frameworks, components, services and applications are all on top of JCA API. The Cryptographic Service Provider (CSP) is in the middle, along with BouncyCastle and Oracle providers. Arithmetic libraries are at the bottom.

Figure 2 shows the CSP divided in two distinct cryptographic libraries. The left side shows only standardized algorithms and comprises a conventional cryptographic library. The right side features only non-standard cryptography and is an alternative library. The following subsections describe these two libraries.

A. *Standard Cryptography*

This subsection details the implementation choices for the standard cryptographic library. The motivations behind this implementation were all characteristics of standardized algorithms: interoperability, documentation, and testability. The standard cryptography is packaged as a pure-Java library according to the JCA specifications.

The programming language chosen for implementation of this cryptographic library was Java. The block cipher is the AES algorithm, which was implemented along with thee of operation: ECB, and CBC [27], as well as the GCM mode for authenticated encryption [28]. PKCS#5 [3] is the simplest padding mechanism and was chosen for compatibility with other CSPs. As GCM mode for authenticated encryption only uses AES encryption, the optimization of encryption received more attention than AES decryption. Implementation aspects of AES and other cryptographic algorithms can be found on the literature [15][24][34], in particular [29].

The asymmetric algorithm is the RSA-PSS that is a Probabilistic Signature Scheme constructed over the RSA signature algorithm. PSS is supposed to be more secure than ordinary RSA [23][34]. Asymmetric encryption is provided by the RSA-OAEP [23][34].

Two cryptographically secure hashes were implemented, SHA-1 [22] and MD5. It is well known by now that MD5 is considered broken and is not to be used in serious applications, it is present for ease of implementation. In current version, there is no intended use for these two hashes. Their primary use will be as the underling hash function in MACs, digital signatures and PRNGs. The Message Authentication Codes chosen were the HMAC [25] with SHA-1 as the underling hash function, and the GMAC [28], which can be directly derived from GCM mode. SHA-2 family of secure hashes supplies the need for direct use of single hashes.

The need for a key agreement was fulfilled by the implementation of Station-to-Station (STS) protocol, which

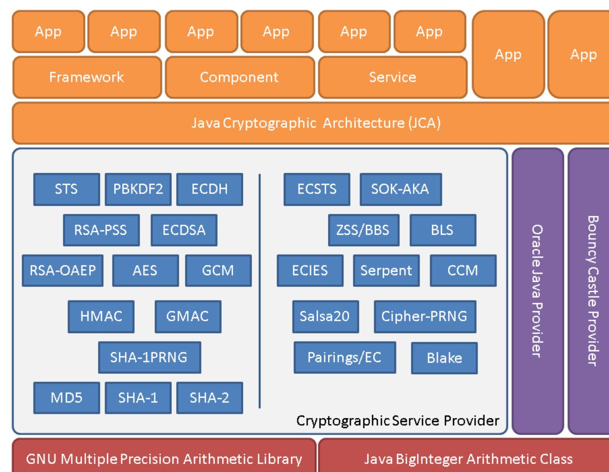

Figure 2. Cryptographic Service Provider Architecture.

is based upon Authenticated Diffie-Hellman [38], and provides mutual key authentication and confirmation [4][39].

Finally, the mechanism for Password-based Encryption (PBE) is based on the Password-Based Key Derivation Function 2 (PBKDF2) [3], and provides a simple and secure way to store keys in encrypted form. In PBE, a key-encryption-key is derived from a password.

B. *Non-standard Cryptography*

This subsection details the implementation choices for the alternative cryptographic library. The non-standard cryptography is a dynamic library written in C and accessible to Java programs through a Java Native Interface (JNI) connector, which acts as a bridge to a JCA adapter.

By the time of writing, this alternative library was under the final steps of its construction. The most advanced cryptographic protocols currently implemented are based upon a reference implementation [5] and are listed below.

a) ECDH [8]. The key agreement protocol ECDH is a variation of the Diffie-Hellman protocol using elliptic curves as the underlying algebraic structure;

b) ECDSA [21]. This is a DSA-based digital signature using elliptic curves. ECSS [8] is a variation of ECDSA that does not require the computation of inverses in the underlying finite field, obtaining a signature algorithm with better performance;

c) SOK [8]. This protocol is a key agreement for Identity-Based Encryption (IBE). Sometimes, it is called SOKAKA for SOK Authenticated Key Agreement;

d) BLS [6]. A short digital signature scheme in which given a message m, it is computed S = H (m), where S is a point on an elliptic curve and H() is a secure hash;

e) ZSS [11]. Similar to the previous case, it is a more efficient short signature, because it utilizes fixed-point multiplication on an elliptic curve rather arbitrary point;

f) Blake [32]. Cryptographic hash function submitted to the worldwide contest for selecting the new SHA-3 standard and was ranked among the five finalists;

g) ECIES [8]. This is an asymmetric encryption algorithm over elliptic curves. This algorithm is non-deterministic

Copyright (c) IARIA, 2014. ISBN: 978-1-61208-376-6

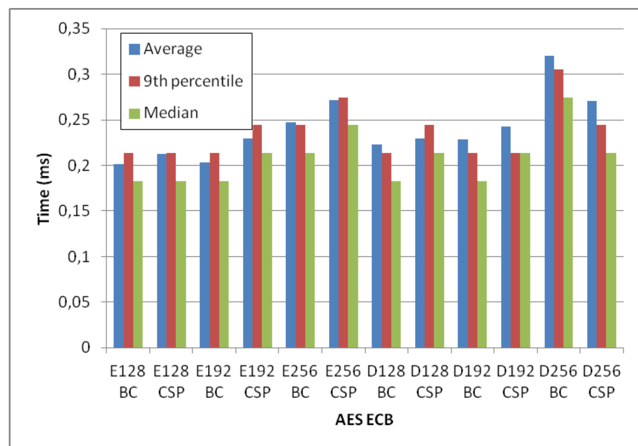

Figure 4. Performance of AES in pure-Java - average, 9th percentile, and median of 10.000 iterations.

and can be used as a substitute of the RSA-OAEP, with the benefit of shorter cryptographic keys;

h) ECSTS [8]. Variation of STS protocol using elliptic curves and ECDH as a replacement for ADH;

i) Salsa20 [9]. This is a family of 256-bit stream ciphers submitted to the ECRYPT Project (eSTREAM);

j) Serpent [31]. A 128-bit block cipher designed to be a candidate to contest that chose the AES. Serpent did not win, but it was the second finalist and enjoys good reputation in the cryptographic community.

C. *Security decisions for non-standard cryptography*

Among the characteristics that were considered in the choice of alternative cryptographic primitives, side channels protection was a prevailing factor and had distinguished role in the design of the library. For instance, schemes with known issues were avoided, while primitives that were constructed to resist against such attacks are currently being regarded for inclusion in the architecture. Furthermore, constant-time programming techniques, like for example in table accessing operations for AES, are being surveyed in order to became part of the implementation.

Concerning mathematical security of non-standard cryptography, the implementation offers alternatives for 256-bit security for both symmetric and asymmetric encryption. For instance, Serpent-256 corresponds to AES-256 block cipher, while the same security level is achieved in asymmetric world using elliptic curves over 521-bit finite fields, what can only be possible in standard cryptography using 15360-bit RSA key size. Thus, in higher security levels, non-standard primitives performance is significantly improved in relation to standard algorithms, but an extensive analysis of this scenario, with concrete timing comparisons, is left as future work.

A final remark about the use of non-standard cryptography is that working with advanced cryptographic techniques that have not been sufficiently analyzed by the scientific community has its own challenges and risks. There are occasions when the design of a non-standard cryptographic library has to be conservative in order to preserve security.

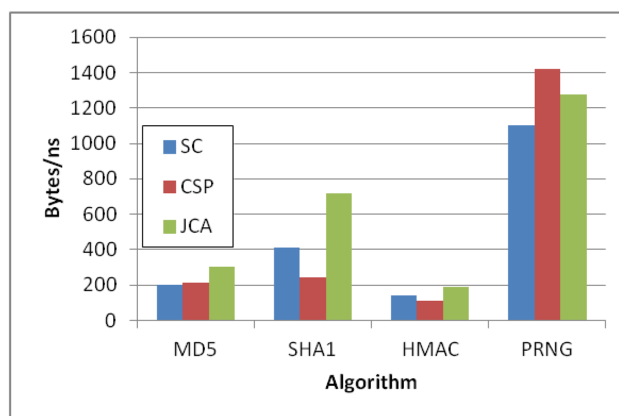

Figure 3. Throughput of implementations.

For instance, a recent improvement in mathematics [12][30] had eliminated an entire line of research in theoretical cryptography. Such advancement affected elliptic curve cryptography using a special kind of binary curves called supersingular curves, but had no effect on the bilinear pairings over primes fields or encryption on ordinary (common) binary curves. Thus, these two technologies remain cryptographically secure. Unfortunately, the compromised curves were in use and had to be eliminated from the cryptographic library.

As pairings on prime fields can still be securely used in cryptographic applications, the implementation was adapted to that new restricted context. Additionally, ordinary elliptic curves may still be used for cryptographic purposes, considering they are not supersingular curves, and the implementation had to adapt to that fact too.

IV. PERFORMANCE EVALUATION

Performance evaluation of Java programs, either in standard JVM or DVM/Android, is a stimulating task due to many sources of interference that can affect measurements. As discussed in previous sections, GC and JiTC have great influence over the performance of Java programs. The intent of performance evaluations presented in this section is to provide and describe a realistic means to compare cryptography implementations in Java.

Two approaches of measurement have been used for the evaluation of cryptographic functions implemented in pure-Java programs. The first one was the measurement of elapsed time for single cryptographic functions processing a single block of data. This approach suffers from the interference of GC and JiTC. The JiTC interference can be eliminated by discarding all the measurements collected before code optimization. The GC interference cannot be completely eliminated, though.

Copyright (c) IARIA, 2014. ISBN: 978-1-61208-376-6

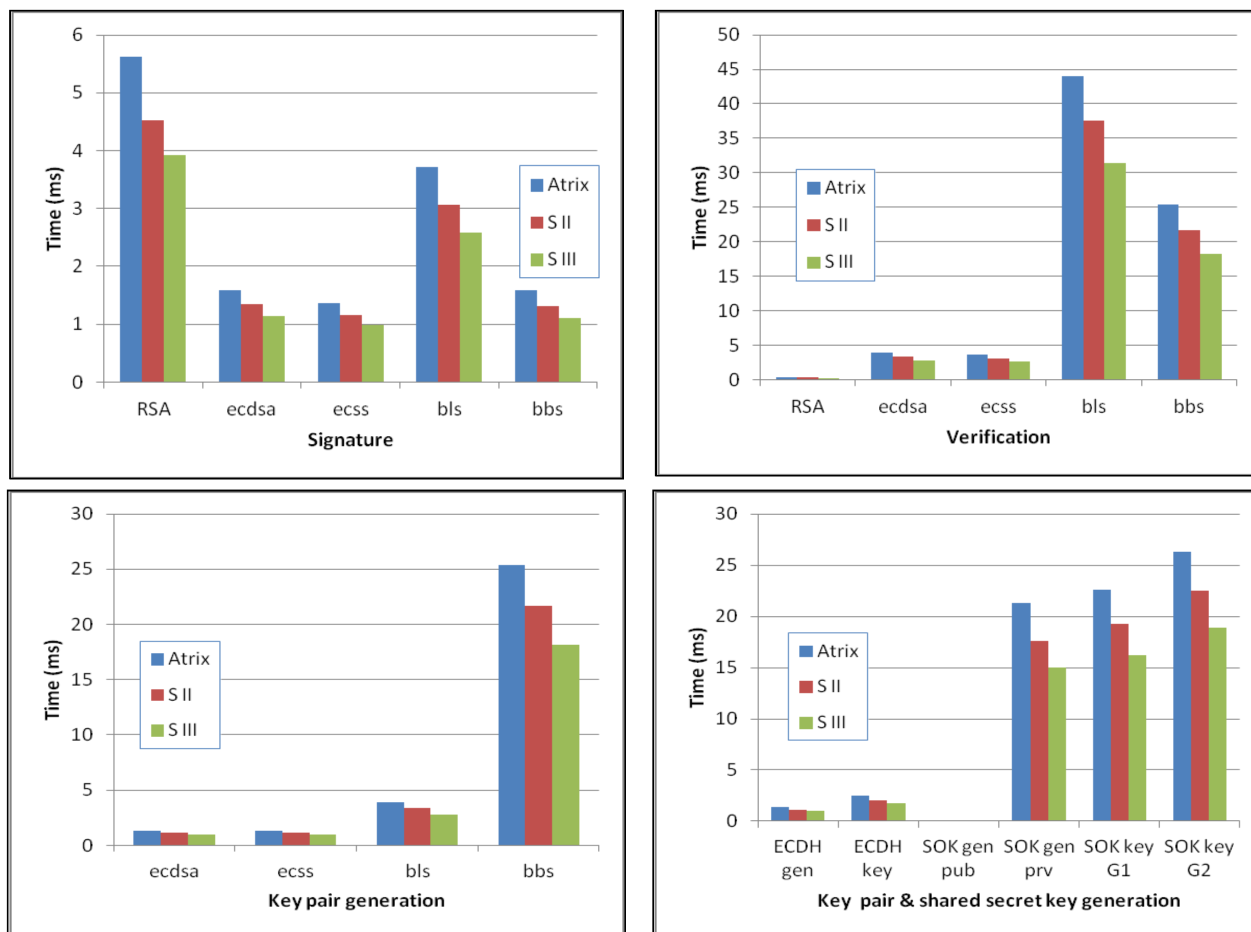

Figure 5. Performance evaluation of non-standard cryptography. Digital signatures: signature generation (top-left) and signature verification (top-right). Key Agreement: key pair generation (bottom-left), secret-key generation (bottom-right).

Figure 4 exemplifies the first approach and shows the comparative performance of AES encryption, in ECB mode, of a single block of data for two Java CSPs: This text CSP and BouncyCastle (BC) [36]. The measurements were taken on an LG Nexus 4 with 16GB of internal storage, 2 GB of RAM, 1.5 GHz Quad-core processor, and Android 4.3. The procedure consisted of processing a single block of data in a loop of 10.000 iterations. AES were setup to three key sizes (128, 192 and 256) in both encryption and decryption.

In order to inhibit the negative influence of GC and JiTC, three metrics were taken: the average of all iterations, the 9th percentile and the median. None of them resulted in a perfect metric. However, these measures do offer a realistic comparison of CSP and BC. They show similar performance.

The second approach for performance evaluation has to consider that final users of mobile devices will not tuning their Java VMs with obscure configuration options in order to achieve maximum performance. On the contrary, almost certainly they will use default configurations, with minor changes on device's settings. Thus, the responsiveness of an application tends to be more relevant to final users than the performance of single operations.

The second approach of measurement takes into account the responsiveness of cryptographic services and considers the velocity with which a huge amount of data can be processed, despite the interferences of GC and JiTC. The amount of work performed per unit of time is called the throughput of the cryptographic implementation.

Figure 3 shows the throughput of four cryptographic services implemented by CSP compared to BC and JCE: MD5, SHA-1, HMAC and SHA1PRGN. The measurements were taken on a smartphone of type Samsung Galaxy S III (Quad-core 1.4 GHz Cortex-A9 processor, 1GB of RAM, and Android 4.1). The procedure consisted of processing an input file of 20 MB, in a loop of 10 iterations. All cryptographic algorithms were setup with a 128-bit key. BouncyCastle has a deployment for Android, called SpongeCastle (SC) [33]. It is interesting to observe that the three CSPs are quite similar in performance.

The previous paragraphs suggest that the pure-Java package of CSP, with standard cryptography only, is quite competitive in performance when compared to other CSP and its use might not be considered a bottleneck to applications.

Copyright (c) IARIA, 2014. ISBN: 978-1-61208-376-6

Unfortunately, the case for non-standard cryptography is not that simple, despite been implemented in C and not been subjected to GC and JiTC influences. Non-standard cryptography usually has no standard specifications or safe reference implementations. Neither it is in broad use by other cryptographic libraries. Because of that, comparisons among implementations of the same algorithm are barely possible. On the other hand, it is feasible to compare alternative and standard cryptography, considering the same type of service.

For the non-standard cryptography implementations, performance measurements were taken in three smartphones: (i) Motorola Atrix with processor of 1 GHz, 1 GB of RAM and 16GB of storage; (ii) Samsung Galaxy S II with processor of 1.2 GHz dual-core ARM Cortex-A9, 1 GB of RAM and 16GB of storage; and (iii) Samsung Galaxy S III with processor of 1.4 GHz quad-core Cortex-A9, 1 GB of RAM, and 16 GB of storage.

Figure 5 shows two types of services: digital signatures at the top and key agreement (KA) at the bottom. The bar chart at top-left quadrant shows generation of digital signatures for five algorithms: RSA, ECDSA, ECSS, BLS and ZSS (BBS). Traditionally, RSA is the slowest one. Elliptic curve cryptography, as in ECDSA and ECSS, is faster. Short signatures, such as BLS and ZSS (BBS), are not as fast as EC.

Bar chart at top-right quadrant shows verification of digital signatures for five algorithms: RSA, ECDSA, ECSS, BLS and ZSS (BBS). Traditionally, RSA verification is the fastest one. Elliptic curve cryptography, as in ECDSA and ECSS, is not that fast. Short signatures, such as BLS and ZSS (BBS), are terribly slow, due to complex arithmetic involved in bilinear pairings computations. The bottom-left quadrant contains a bar chart showing key pair generation for ECDSA, ECSS, BLS, and ZSS (BBS). Again, performance is slow for BLS and ZSS (BBS) due to complex arithmetic involved in bilinear pairings.

Bar chart in bottom-right quadrant shows operations for two KA schemes: ECDH and SOK. ECDH is quite fast in generating parameters (both public and private), as well as in generating the shared secret. But, pairings based KA schemes are relatively slow in both operations.

V. CONCLUDING REMARKS

This paper discussed implementation issues on the construction of a cryptographic library for Android smartphones. The library actually consists of both standard and non-standard cryptographic algorithms. Performance measurements were taken in order to compare CSP with other cryptographic providers. Despite all difficulties for obtain realistic data, experiments have shown that standard CSP can be competitive to other implementations. On the other hand, non-standard cryptography has shown low performance that can possibly inhibit its use in real time applications. However, their value consists in offering secure alternatives to possibly compromised standards. Future work will focus on correctness, security (particularly in the context of side channel attacks) and performance optimization. Correctness of implementation in the absence of formal verification is a primary concern and should be taken seriously, particularly for non-standard cryptography.

Finally, regarding recent global surveillance disclosures, non-standard cryptographic primitives can be faced as part of the usual trade-offs that directs the design of cryptographically secure applications.

ACKNOWLEDGMENT

The authors acknowledge the financial support given to this work, under the project "Security Technologies for Mobile Environments – TSAM", granted by the Fund for Technological Development of Telecommunications – FUNTTEL – of the Brazilian Ministry of Communications, through Agreement Nr. 01.11. 0028.00 with the Financier of Studies and Projects - FINEP / MCTI.

REFERENCES

[1] A. Braga and E. Nascimento, Portability evaluation of cryptographic libraries on android smartphones. In Proceedings of the 4th international conference on Cyberspace Safety and Security (CSS'12), Yang Xiang, Javier Lopez, C.-C. Jay Kuo, and Wanlei Zhou (Eds.). Springer-Verlag, Berlin, Heidelberg, 2012, pp. 459-469.

[2] A. Braga, Integrated Technologies for Communication Security on Mobile Devices. In MOBILITY, The Third International Conference on Mobile Services, Resources, and Users, 2013, pp. 47-51.

[3] B. Kaliski, RFC 2898. PKCS #5: Password-Based Cryptography Specification Version 2.0. Available in: http://tools.ietf.org/html/rfc2898.

[4] B. O'Higgins and W. Diffie and L. Strawczynski, R. do Hoog, Encryption and ISDN - A Natural Fit, 1987 International Switching Symposium (ISS87), 1987.

[5] D. Aranha and C. Gouvêa, RELIC, RELIC is an Efficient LIbrary for Cryptography, Available in: http://code.google.com/p/relic-toolkit.

[6] D. Boneh and B. Lynn and H. Shacham, Short signatures from the Weil pairing. J. Cryptology, Extended abstract in Proceedings of Asiacrypt 2001, Sept. 2004, 17(4): pp. 297–319.

[7] D. Bornstain, Dalvik, VM Internals. Available in: http://sites.google.com/site/io/dalvik-vm-internals.

[8] D. Hankerson and S. Vanstone and A. Menezes, Guide to elliptic curve cryptography, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[9] D. J. Bernstein, The Salsa20 family of stream ciphers. Available in: http://cr.yp.to/papers.html#salsafamily.

[10] Ergonomics in the 5.0 Java Virtual Machine. Available in: http://www.oracle.com/technetwork/java/ergo5-140223.html

[11] F. Zhang and R. Safavi-Nainia and W. Susilo, An Efficient Signature Scheme from Bilinear Pairings and Its Applications., in F. Bao and R. H. Deng and J. Zhou, ed., Public Key Cryptography, Springer, 2004, pp. 277-290.

[12] G. Anthes, “French team invents faster code-breaking algorithm”, Communications of the ACM, v. 57,n. 1, January 2014, pp. 21-23.

[13] H. Oh and B. Kim and H. Choi and S. Moon, Evaluation of Android Dalvik virtual machine. In Proceedings of the 10th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES '12), ACM, New York, NY, USA, 2012, pp. 115-124.

[14] How to Implement a Provider in the Java Cryptography Architecture. Available in: http://docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/HowToImplAProvider.html.

[15] J. Bos and D. Osvik and D. Stefan, Fast Implementations of AES on Various Platforms, 2009. Available in http://eprint.iacr.org/2009/501.pdf.

[16] J. Menn, Experts report potential software "back doors" in U.S. standards. Available in: http://www.reuters.com/article /2014/07/15/usa-nsa-software-idUSL2N0PP2BM20140715?irpc=932.

[17] Java Cryptography Architecture Oracle Providers Documentation for Java Platform Standard Edition 7. Available in: http://docs.oracle.com/javase/7/docs/technotes/guides/security/SunProviders.html.

[18] Java Cryptography Architecture Standard Algorithm Name Documentation for Java Platform Standard Edition 7. Available in: http://docs.oracle.com/javase/7/docs/technotes/guides/security/StandardNames.html.

[19] Java Cryptography Extension Unlimited Strength Jurisdiction Policy Files 7 Download. Available in: http://www.oracle.com/technetwork/pt/java/javase/downloads/jce-7-download-432124.html.

[20] Java™ Cryptography Architecture (JCA) Reference Guide. Available in: http://docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/CryptoSpec.html.

[21] NIST FIPS PUB 186-2, Digital Signature Standard (DSS). Available in: http://csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf.

[22] NIST FIPS-PUB-180-4. Secure Hash Standard (SHS). Available in: http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf, March 2012.

[23] NIST FIPS-PUB-186. Digital Signature Standard (DSS). Available in: http://csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf.

[24] NIST FIPS-PUB-197. Announcing the ADVANCED ENCRYPTION STANDARD (AES). Federal Information Processing Standards Publication 197 November 26, 2001.

[25] NIST FIPS-PUB-198. The Keyed-Hash Message Authentication Code (HMAC). Available in: http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf.

[26] NIST Removes Cryptography Algorithm from Random Number Generator Recommendations. Available in: http://www.nist.gov/itl/csd/sp800-90-042114.cfm.

[27] NIST SP 800-38A. Recommendation for Block Cipher Modes of Operation. 2001. Available in: http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf.

[28] NIST SP 800-38D. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. 2007. Available in: http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf.

[29] Paulo Barreto's AES Public Domain Implementation in Java. Available in: www.larc.usp.br/~pbarreto/JAES.zip.

[30] R. Barbulescu, P. Gaudrey, A. Joux,, and E. Thomé, "A quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic", June 2013, preprint available at http://eprint.iacr.org/2013/400.pdf.

[31] SERPENT, A Candidate Block Cipher for the Advanced Encryption Standard. Available in: www.cl.cam.ac.uk/~rja14/serpent.html.

[32] SHA-3 proposal BLAKE. Available in: https://131002.net/blake.

[33] SpongyCastle, Spongy Castle: Repackage of Bouncy Castle for Android, Bouncy Castle Project. Available in: http://rtyley.github.com/spongycastle/, 2012.

[34] T. St. Denis and S. Johnson, Cryptography for Developers. Syngress, 2006.

[35] The Java HotSpot Performance Engine Architecture. Available in: www.oracle.com/technetwork/java/whitepaper-135217.html.

[36] The Legion of the Bouncy Castle. Legion of the Bouncy Castle Java cryptography APIs. Available in: www.bouncycastle.org/java.html.

[37] Tuning Garbage Collection with the 5.0 Java Virtual Machine. Available in: http://www.oracle.com/technetwork/ java/gc-tuning-5-138395.html.

[38] W. Diffie and M. Hellman, New Directions in Cryptography, IEEE Trans. on Inf. Theory, vol. 22, no. 6, Nov. 1976, pp. 644-654.

[39] W. Diffie and P. C. van Oorschot, M. J. Wiener, Authentication and Authenticated Key Exchanges, Designs, Codes and Cryptography (Kluwer Academic Publishers) 1992, 2 (2): pp. 107–125.