Integrated Technologies for Communication Security and Secure Deletion on Android Smartphones

Alexandre Melo Braga, Daniela Castilho Schwab, Eduardo Moraes de Morais, Romulo Zanco Neto, and André Luiz Vannucci
Centro de Pesquisa e Desenvolvimento em Telecomunicações (Fundação CPqD)
Campinas, São Paulo, Brazil
{ambraga,dschwab,emorais,romulozn,vannucci}@cpqd.com.br

Abstract— **Nowadays, mobile devices are powerful enough to accomplish most of the tasks previously accomplished only by personal computers; that includes, for example, file management and instant messaging. On the other hand, in order to protect final user's interests, there is also an increasing need for security hardenings on ordinary, off-the-shelf devices. In fact, there is a need for practical security technologies that work at the application level, above the operating system and under the control of the user. This technology has to be easy to use in everyday activities and easily integrated into mobile devices with minimal maintenance and installation costs. The main contribution of this paper is to describe design and implementation issues concerning the construction of an integrated framework for securing both communication and storage of sensitive information of Android smartphones. Four aspects of the framework are detailed in this paper: the construction of a cryptographic library, its use in the development of a cryptographically secure instant message service, the integration with an encrypted file system, and the addition of secure deletion technologies. Also, an analysis of non-standard cryptography is provided, as well as performance evaluation of a novel secure deletion technique. The proposed framework is supposed to work in user-mode, as an ordinary group of mobile apps, without root access, with no need for operating system modification, in everyday devices.**

Keywords-cryptography; surveillance; security; Android; instant message; secure deletion; secure storage; encrypted file system; flash memory.

I. INTRODUCTION

Nowadays, the proliferation of smartphones and tablets and the advent of cloud computing are changing the way people handle their personal, maybe private, information. In fact, many users keep their sensitive data on mobile devices as well as on cloud servers.

The current generation of mobile devices is powerful enough to accomplish most of the tasks previously accomplished only by personal computers. That includes, for example, file management operations (such as create, read, update, and delete) and instant message capabilities. Also, today's devices possess operating systems that are hardware-agnostic by design and abstract from ordinary users all hardware details, such as writing procedures for flash memory cards.

However, there is no free lunch, and mobile devices, as any other on-line computer system, are vulnerable to many kinds of data leakage. Unfortunately, as the amount of digital data in mobile devices grows, so does the theft of sensitive data through loss of the device, exploitation of vulnerabilities or misplaced security controls. Sensitive data may also be leaked accidentally due to improper disposal of devices.

Contemporary to this paradigm shift from ordinary computers to mobile devices, the use in software systems of security functions based on cryptographic techniques seems to be increasing as well, maybe as a response to the new security landscape. The scale of cryptography-based security in use today seems to have increased not only in terms of volume of encrypted data, but also relating to the amount of applications with cryptographic services incorporated within their functionalities. In addition to the traditional use cases historically associated to stand-alone cryptography (e.g., encryption/decryption and signing/verification), there are new application-specific usages bringing diversity to the otherwise known threats to cryptographic software.

For example, today's secure phone communication does not mean only voice encryption, but encompasses a plethora of security services built over the ordinary smartphone capabilities. To name just a few of them, these are SMS encryption, Instant Message (IM) encryption, voice and video chat encryption, secure conferencing, secure file transfer, secure data storage, secure application containment, and remote security management on the device, including management of cryptographic keys. It is not surprisingly that, with the increasing use of encryption systems, an attacker wishing to gain access to sensitive data is directed to weaker targets. On mobile devices, one such attack is the recovery of supposedly erased data from internal storage, possibly a flash memory card. Also, embedded security technologies can suffer from backdoors or inaccurate implementations, in an attempt to facilitate unauthorized access to supposedly secure data.

This paper describes design and implementation issues concerning the construction of an integrated framework for securing both communication and storage of sensitive information of Android smartphones. Preliminary versions of this work have been addressed in previous publications [1][2][3], as part of a research project [4][5][6] targeting security technologies on off-the-shelf mobile devices.

Additionally, it is a real threat the misuse of security standards by intelligence agencies. The motivation behind the special attention given to the selection of cryptographic algorithms lies in the recently revealed weakness, which may be intentionally included by foreign intelligence agencies, in international encryption standards [7][8]. 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 cryptography.

This work contributes to the state of the practice by discussing the technical aspects and challenges of cryptographic implementations, as well as their integration into security-ware applications on modern, Android-based, mobile devices. The main contributions of this paper are the following:

- Discuss the construction of a cryptographic library for Android devices, which focuses on design decisions as well as on implementation issues of both standard and non-standard algorithms;
- Describe the construction of a mobile application for secure instant messaging that uses the cryptographic library and is integrated with an encrypted file system;
- Describe an encrypted file-system that uses the cryptographic library and integrates secure deletion technologies;
- Propose and analyze new approaches to secure deletion of stored data on off-the-shelf mobile devices.

The remaining parts of the text are organized as follows. Section II offers background on the subject. Section III presents related work. Section IV treats the construction of the secure chat. Section V details the constructions of the cryptographic library. Section VI describes the encrypted file system with secure deletion. Section VII discusses integration aspects. Section VIII concludes this text.

II. BACKGROUND

This section offers background information in the following selected subjects of interest: Android and Java technologies; cryptography issues in mobile devices; and secure storage and data deletion in flash memories.

A. General concepts for Android and Java

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

1) 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 [9] is a software framework for use and development of cryptographic primitives in the Java platform. JCA defines, among other facilities, Application Program Interfaces (APIs) for digital signatures and secure hash functions [9]. On the other hand, APIs for encryption, key establishment and message authentication codes (MACs) are defined in the Java Cryptography Extension (JCE) [10].

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 [11].

2) Garbage Colletion and JiT Compilation

An architectural feature of the JVM has great influence in the general performance of applications: the GC [12][13]. Applications have different requirements for 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. 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.

Another important consideration on performance of Java programs is the JiT Compilation [12][14]. Historically, Java bytecode used to be fully interpreted by the JVM and presented serious performance issues. Nowadays, JiTC not only compiles Java programs, but also optimizes 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.

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. Further discussion of cryptographic side-channels and its detection in Java can be found in [15].

3) DVM

The DVM [16] 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 [17]. 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 [17] 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.

B. Security and cryptography issues on Android devices

A broad study on Android application security, especially focused on program decompilation and source code analysis, was performed by [18]. There are several misuse commonly found on cryptographic software in use today. According to a recent study [19], the most common misuse of cryptography in mobile devices is the use of deterministic encryption, where a symmetric cipher in Electronic Code Book (ECB) mode appears mainly in two circumstances: Advanced Encryption Standard (AES) in ECB mode of operation (AES/ECB for short) and Triple Data Encryption Standard in ECB mode (TDES/ECB). A possibly worse variation of this misuse is the Rivest-Shamir-Adleman (RSA) cryptosystem in Cipher-Block Chaining (CBC) mode with Public-Key Cryptography Standards Five (PKCS#5) padding (without randomization) [20]. Another frequent misuse is hardcoded Initialization Vectors (IVs), even with fixed or constant values [20]. A related misuse is the bad habit of hardcoded seeds for PRNGs [19].

A common misunderstanding concerning the correct use of IVs arises when (for whatever reason) programmers need to change operation modes of block ciphers. For instance, the Java Cryptographic API [9] allows operation modes to be easily changed, but without considering IV requirements.

According to a NIST standard [21], CBC and Cipher feedback (CFB) modes require unpredictable IVs. However, Output feedback (OFB) mode does not need unpredictable IVs, but it must be unique to each execution of the encryption operation. Considering these restrictions, IVs must be both unique and unpredictable, in order to work interchangeably with almost all common operation modes of block ciphers. The Counter (CTR) mode requires unique IVs and this constraint is inherited by authenticated encryption with Galois/Counter mode (GCM) [22].

C. Secure storage and deletion on flash memory

Traditionally, the importance of secure deletion is well understood by almost everyone and several real-world examples can be given on the subject: sensitive mail is shredded; published government information is selectively redacted; access to top secret documents ensures all copies can be destroyed; and blackboards at meeting rooms are erased after sensitive appointments.

In mobile devices, that metaphor is not easily implemented. All modern file systems allow users to “delete” their files. However, on many devices the remove-file command misleads the user into thinking that her file has been permanently removed, when that is not the case. File deletion is usually implemented by unlinking files, which only changes file system metadata to indicate that the file is “deleted”; while the file’s full content remains available in physical medium. This process is known as logical deletion.

Unfortunately, despite the fact that deleted data are not actually destroyed in the device, logical deletion has the additional drawback that ordinary users are generally unable to completely remove her files. On the other hand, advanced users or adversaries can easily recover logically deleted files.

Deleting a file from a storage medium serves two purposes: (i) it reclaims storage to operating system and (ii) ensures that any sensitive information contained in the file becomes inaccessible. The second purpose requires that files are securely deleted.

Secure data deletion can be defined as the task of deleting data from a physical medium so that the data is irrecoverable. That means its content does not persist on the storage medium after the secure deletion operation.

Secure deletion enables users to protect the confidentiality of their data if their device is logically compromised (e.g., hacked) or stolen. Until recently, the only user-level deletion solution available for mobile devices was the factory reset, which deletes all user data on the device by returning it to its initial state. However, the assurance or security of such a deletion cannot be taken for granted, as it is highly dependent on device’s manufacturer. Also, it is inappropriate for users who wish to selectively delete data, such as some files, but still retain their address books, emails and installed applications.

Older technologies [23] claim to securely delete files by overwriting them with random data. However, due the nature of log-structured file systems used by most flash cards, this solution is no more effective than logically deleting the file, since the new copy invalidates the old one but does not physically overwrite it. Old secure deletion approaches that work at the granularity of a file are inadequate for mobile devices with flash memory cards.

Today, secure deletion is not only useful before discarding a device. On modern mobile devices, sensitive data can be compromised at unexpected times by adversaries capable of obtaining unauthorized access to it. Therefore, sensitive data should be securely deleted in a timely fashion.

Secure deletion approaches that target sensitive files, in the few cases where it is appropriate, must also address usability concerns. A user should be able to reliably mark their data as sensitive and subject to secure deletion. That is exactly the case when a file is securely removed from an encrypted file system. On the other hand, approaches that securely delete all logically deleted data, while less efficient, suffer no false negatives. That is the case for purging.

III. RELATED WORK

This section discusses related work on following subjects: cryptography implementation on mobile devices, security of IM applications, and secure storage and deletion.

A. Cryptography implementation on mobile devices

A couple of years ago, the U.S. National Security Agency (NSA) started to encourage the use of off-the-shelf mobile devices, in particular smartphones with Android, for communication of classified information [24]. The document fosters the adoption of two layers of cryptography for communication security. One is provided by infrastructure (e.g., VPN) and other implemented at the application layer.

Regarding the performance evaluation of cryptographic libraries on Android smartphones, there are tests made on the Android platform for the BouncyCastle and Harmony cryptographic libraries, both already available on the platform [25].

A few works could be found concerning efficient implementation of cryptography on smartphones. The first one [26] presented an efficient Java implementation of elliptic curve cryptography for J2ME-enabled mobile devices. That Java implementation has an optimized scalar multiplication that combines efficient finite-field arithmetic with efficient group arithmetic. A second work [27] presented an identity-based key agreement protocol for securing mobile telephony in GSM and UMTS networks. The paper proposes an approach to speed up client-side cryptography using server-aided cryptography, by outsourcing computationally expensive cryptographic operations to a high-performance backend computing server.

Another work [28] presents a Java port (jPBC) of the PBC library written in C, which provides simplified use of bilinear maps and supports different types of elliptic curves.

A recent study [6] showed that despite the observed diversity of cryptographic libraries in academic literature, this does not mean those implementations are publicly available or ready for integration with third party software. In spite of many claims on generality, almost all of them were constructed with a narrow scope in mind and prioritizes academic interest for non-standard cryptography. Furthermore, portability to Android used to be a commonly neglected concern on cryptographic libraries [6].

Recently, the European Union Agency for Network and Information Security (ENISA) has published two technical reports [29][30] about the correct and safe use of cryptography to protect private data in on-line system, giving attention to cloud and mobile environments. One report [29] focuses on algorithms, key size and parameters. Other report [30] gives attention to cryptographic protocols, and tries to point legacy issues and design vulnerabilities.

B. Security issues in IM protocols and applications

The work of Xuefu and Ming [31] shows the use of eXtensible Messaging and Presence Protocol (XMPP) for IM on web and smartphones. Massandy and Munir [32] have done experiments on security aspects of communication, but there are unsolved issues, such as strong authentication, secure storage, and implementation of good cryptography, as shown by Schrittwieser et al. [33].

It seems that the most popular protocol for secure IM in use today is the Off-the-Record (OTR) Messaging [34], as it is used by several secure IM apps. OTR Messaging handshake is based upon the SIGMA key exchange protocol [35], a variant of Authenticated Diffie-Hellman (ADH) [36], just like Station-to-Station (STS) [37][38].

A good example of security issues found in current IM software is a recently discovered vulnerability in WhatsApp [39]. The vulnerability resulting from misuse of the Rivest Cipher 4 (RC4) stream cipher in a secure communication protocol allowed the decryption, by a malicious third party able to observe conversations, of encrypted messages exchanged between two WhatsApp users.

In order to be fair, it is worth note that WhatsApp has recently announced an effort for hardening its communication security with end-to-end encryption [40].

C. Secure storage and deletion

This section briefly describes related work on the subjects of secure deletion and encrypted file systems on mobile devices, particularly Android.

The use of cryptography as a mechanism to securely delete files was first discussed by Boneh and Lipton [41]. Their paper presented a system which enables a user to remove a file from both file system and backup tapes on which the file is stored, just by forgetting the key used to encrypt the file.

Gutman [23] covered methods available to recover erased data and presented actual solutions to make the recovery from magnetic media significantly more difficult by an adversary. Flash memory barely existed at the time it was written, so it was not considered by him.

K. Sun et al. [42] proposed an efficient secure deletion scheme for flash memory storage. This solution resides inside the operating system and close to the memory card controller.

Diesburg and Wang [43] presented a survey summarizing and comparing existing methods of providing confidential storage and deletion of data in personal computing environments, including flash memory issues.

Wang et al. [44] present a FUSE (File-system in USErspace) encryption file system to protect both removable and persistent storage on devices running the Android platform. They concluded that the encryption engine was easily portable to any Android device and the overhead due to encryption is an acceptable trade-off for achieving the confidentiality requirement.

Reardon et al. [45]-[49] have shown plenty of results concerning both encrypted file system and secure deletion. First, Reardon et al. [45] proposed the Data Node Encrypted File System (DNEFS), which uses on-the-fly encryption and decryption of file system data nodes to efficiently and securely delete data on flash memory systems. DNEFS is a modification of existing flash file systems or controllers that extended a Linux implementation and was integrated in Android operating system, running on a Google Nexus One smartphone.

Reardon et al. [46] also propose user-level solutions for secure deletion in log-structured file systems: purging, which provides guaranteed time-bounded deletion of all data previously marked to be deleted, and ballooning, which continuously reduces the expected time that any piece of deleted data remains on the medium. The solutions empower users to ensure the secure deletion of their data without relying on the manufacturer to provide this functionality. These solutions were implemented on an Android smartphone (Nexus One).

In two recent papers, Reardon et al. [47][48] study the issue of secure deletion in details. First, in [47], they identify ways to classify different approaches to securely deleting data. They also describe adversaries that differ in their capabilities, show how secure deletion approaches can be integrated into systems at different interface layers. Second, in [48], they survey the related work in detail and organize existing approaches in terms of their interfaces to physical media. More recently, Reardon et al. [49] presented a general approach to the design and analysis of secure deletion for persistent storage that relies on encryption and key wrapping.

Finally, Skillen and Mannan [50] designed and implemented a system called Mobiflage that enables plausibly deniable encryption (PDE) on mobile devices by hiding encrypted volumes within random data on a device's external storage. They also provide [51] two different implementations for the Android OS to assess the feasibility and performance of Mobiflage: One for removable SD cards and other for internal partition for both apps and user accessible data.

The above mentioned works suffer from at last one of the following disadvantages:

- Requires modification of the host operating system or device, so the solution does not work on off-the-shelf devices without modification of OS internals;
- Limits the available (free) storage to ordinary applications, possibly leading apps to starvation by lack of storage;
- Inserts abnormal behavior to storage usage that can potentially slow down the whole system, when using incremental memory sweeping by a single-file, single-thread application.

The secure deletion approach proposed in this paper provides alternative solutions to these disadvantages.

IV. CONSTRUCTION OF A SECURE CHAT APPLICATION

This section describes design and implementation issues concerning the construction of CryptoIM, a prototype app for cryptographically secure, end-to-end communication, which operates on a device-to-device basis, exchanging encrypted instant messages via standard transport protocols.

A. *Cryptographic services for IM applications*

CryptoIM implements the basic architecture used by all IM applications, using the standard protocol XMPP [52] at the transport layer. The application then adds a security layer to XMPP, which is composed of a protocol for session key agreement and cryptographic transaction to transport encrypted messages. The security negotiation is indeed a protocol for key agreement, as illustrated by Figure 1.

To accomplish cryptographically secure communication, Alice and Bob agree on the following general requirements:

- An authentication mechanism of individual messages;
- An encryption algorithm and modes of operation;
- A key agreement protocol;

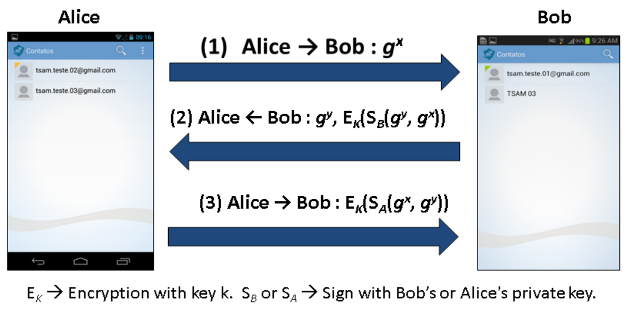

Alice
Bob
(1) Alice → Bob : g^x
(2) Alice ← Bob : g^y, $E_K(S_B(g^y, g^x))$
(3) Alice → Bob : $E_K(S_A(g^x, g^y))$
E_K → Encryption with key k. S_B or S_A → Sign with Bob's or Alice's private key.

Figure 1. Station to Station (STS) protocol.

- A mechanism to protect cryptographic keys at rest.

To avoid known security issues in instant messaging applications [33][39], the key agreement protocol must provide the following security properties [53]:

- Mutual authentication of entities;
- Mutually authenticated key agreement;
- Mutual confirmation of secret possession;
- Perfect Forward Secrecy (PFS).

As a general goal, the CryptoIM is intended to be used in the protection of cryptographically secure communication via mobile devices. In order to be useful, the underlying cryptographic library had to accomplish a minimum set of functional requirements.

Once JCA [9] was defined as the architectural framework, as it is the standard API for cryptographic services on Android, the next design decision was to choose the algorithms minimally necessary to implement a scenario of secure communication via mobile devices. The choice of a minimalist set was an important design decision in order to provide a fully functional Cryptographic Service Provider (CSP) in a relatively short period of time. This minimalist construction had to provide the following set of functions:

a) A symmetric algorithm to be used as block cipher, along with the corresponding key generation function, and modes of operation and padding;
b) An asymmetric algorithm for digital signatures, along with the key-pair generation function. This requirement brings with it the need for some sort of digital certification of public keys;
c) A one-way secure hash function. This is a support function to be used in MACs and signatures;
d) A Message Authentication Code (MAC), based on a secure hash or on a block cipher;
e) A key agreement mechanism or protocol to be used by communicating parties that have never met before, but need to share an authentic secret key;
f) A simple way to keep keys safe at rest and that does not depend on hardware features;
g) A Pseudo-Random Number Generator (PRNG) to be used by key generators and nonce generators.

The cryptographic library supporting CryptoIM was designed to meet each one of these general requirements, resulting in an extensive implementation.

B. *Advanced cryptographic features*

Three improvements to CryptoIM were necessary to integrate it to other apps in the framework. The first is a mobile PKI for digital certification, which is fully integrated to the mobile security framework. PKI's Server-side is based upon the EJBCA [54]. Client-side follows recommendations for handling certificates on mobile devices [55].

The second is a secure text conference (or group chat) via instant messages. As depicted in Figure 2, the Organizer or Chair of the conference requests the conference creation to the Server, as this is an ordinary XMPP feature. The key agreement for the requested conference proceeds as follows, where $Enc_k(x)$ means encryption of x with key k:

1. Chair (C) creates the key for that conference (c_k);
2. For each guest (g[i]), Chair (C) does:

   a) Opens a STS channel with key k: C $\leftrightarrow$ g[i], key k;

   b) Sends c_k on time t to g[i]: C $\rightarrow$ g[i]: $Enc_k(c_k)$.

These steps constitute a point-to-point key transport using symmetric encryption, which is carried out by STS protocol. After that, all guests share the same group key and conference proceeds as a multicast of encrypted messages.

The third improvement is a secure file transfer that is fully integrated to the encrypted file system described in Section VI. Figure 3 illustrated the secure transfer as a step-by-step procedure. The encrypted file system and its file management tool are jointly referred as CryptoFM. The eleven steps for secure file transfer are as follows:

1. Alice activates the file transfer function;
2. Alice's CryptoIM activates the local instance of CryptoFM and passes to it the key K_{FT} (key derived from K_{STS} conversation) for secure transport of files;

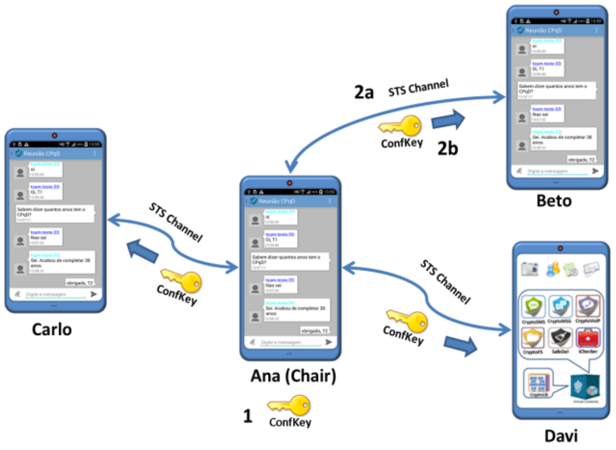


Figure 2. Key agreement for secure conference.

3. Alice chooses, from her CryptoFM, the file to be transferred and exports it from encrypted file system;
4. The exported file is encrypted with the key K_{FT} and stored in a public folder;
5. CryptoIM gets the encrypted file from public folder;
6. The encrypted file and related metadata are transmitted from Alice to Bob through a secure channel (STS channel) over XMPP;
7. The file is received by Bob, who accepts the transfer in his CryptoIM and saves the file;
8. The encrypted file is temporarily saved in a public folder recognized by the Bob's CryptoFM;
9. Bob's CryptoIM activates its local CryptoFM and passes to it the key K_{FT} (key derived from K_{STS}

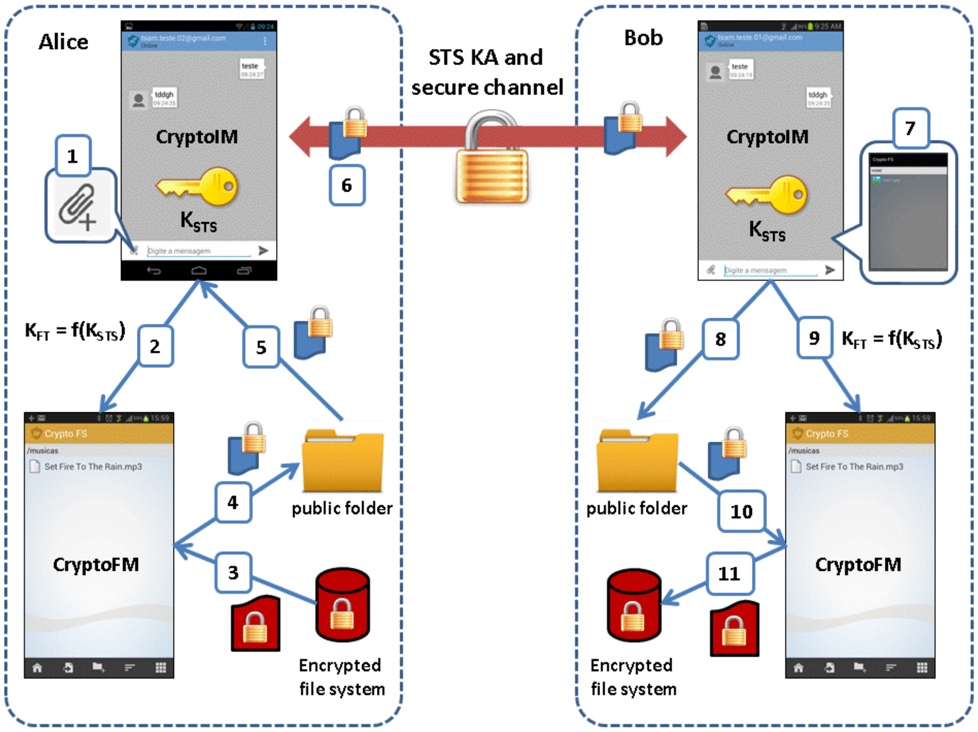


Figure 3. Secure file transfer is integrated to both CryptoIM and CryptoFM.

conversation) used to securely transport of the file;

10. Bob's CryptoFM, in a secure import operation, gets the encrypted file from the public folder and decrypts with key K_{FT};
11. CryptoFM saves the received file into its encrypted file system.

This procedure has the possible vulnerability of leaving temporary files or residual, unencrypted information at local storage. This vulnerability can show up at both sides of file transfer. In fact, this issue raised the need for a method for secure deletion and memory purging.

In summary, the three remarkable differences between CryptoIM and the related work are the following. First, the prototype uses STS protocol and its variants to accomplish authenticated key agreement. This has the benefit of facilitating protocol extension to use alternative cryptographic primitives. Also, STS is used as building block for both multi-user conference and secure file transfer. Second, authenticated encryption is the preferred encryption mechanism to protect messages, so the burden of IV management is minimized. Third, it is fully integrated to an encrypted file system.

V. Construction of a Cryptographic Library

This section describes both the design decisions and implementation issues concerning the construction of a cryptographic library for Android devices. This library support all secure apps included in the secure framework, including CryptoIM, a secure chat detailed in Section IV, and CryptoFM, an encrypted file-system introduced in Section IV and detailed in Section VI.

Four aspects of the implementation were discussed in this paper: selection of cryptographic primitives, architecture of components, performance evaluation on Android devices, and the implementation of non-standard cryptographic algorithms.

As previously stated, 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.

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 [9][56], its name conventions [57], and design principles [10][58].

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 4 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

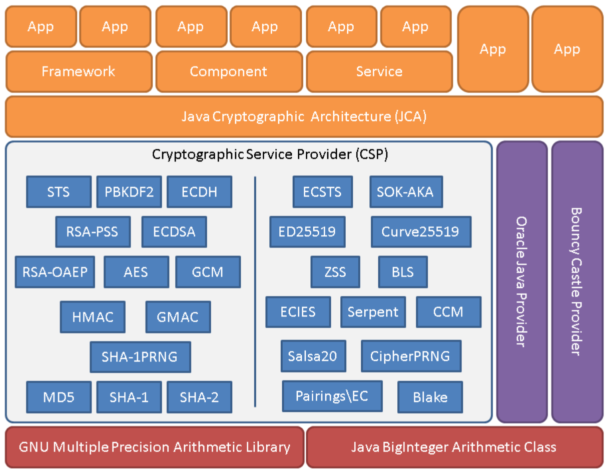


Figure 4. Cryptographic Service Provider Architecture.

BouncyCastle and Oracle providers. Arithmetic libraries are at the bottom.

Figure 4 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 block cipher is the AES algorithm, which was implemented along with the modes of operation: ECB, and CBC [21], as well as the GCM mode for authenticated encryption [22]. PKCS#5 [59] 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 literature [60][61][62], in particular [63].

The Signature algorithm is the RSA-PSS that is a Probabilistic Signature Scheme (PSS) constructed over the RSA signature algorithm. RSA-PSS is supposed to be more secure than ordinary RSA [62][64]. Asymmetric encryption is provided by the RSA-OAEP [62][64].

Two cryptographically secure hashes were implemented, SHA-1 [65] 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 [66] with SHA-1 and SHA2 as the underling hash functions, and the

GMAC [22], 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 key agreement was fulfilled by the Station-to-Station (STS) protocol, which is based upon Authenticated Diffie-Hellman [36], and provides mutual key authentication and confirmation [37][38].

Finally, the mechanism for Password-based Encryption (PBE) is based on the Password-Based Key Derivation Function 2 (PBKDF2) [59], 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.

Some of the constructs are based upon a reference implementation [67]. The most advanced cryptographic protocols currently implemented are listed below:

a) Curve25519 [68] is used to provide a key agreement protocol equivalent to the Elliptic Curve Diffie–Hellman (ECDH) [69], but over a non-standard curve. The key agreement protocol ECDH is a variation of the Diffie-Hellman (DH) protocol using elliptic curves as the underlying algebraic structure;
b) ED25519 [70] is utilized to construct a digital signature scheme that corresponds to the Elliptic Curve Digital Signature Algorithm (ECDSA) [71], but over a non-standard curve that is birationally equivalent to Curve25519. ECSS [69] 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) Sakai-Ohgishi-Kasahara (SOK) [72]. This protocol is a key agreement for Identity-Based Encryption (IBE). Sometimes, it is called SOKAKA for SOK Authenticated Key Agreement;
d) Boneh-Lynn-Shacham (BLS) [73]. 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) Zhang-Safavi-Susilo (ZSS) [74]. 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 [75]. Cryptographic hash function submitted to the worldwide contest for selecting the new SHA-3 standard and was ranked among the five finalists;
g) Elliptic Curve Integrated Encryption Scheme (ECIES) [69]. This is an asymmetric encryption algorithm over elliptic curves. This algorithm is non-deterministic and can be used as a substitute of the RSA-OAEP, with the benefit of shorter cryptographic keys;
h) Elliptic Curve Station-to-Station (ECSTS) [69]. Variation of STS protocol using elliptic curves and ECDH as a replacement for ADH;
i) Salsa20 [76]. This is a family of 256-bit stream ciphers submitted to the ECRYPT Project (eSTREAM);
j) Serpent [77]. 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;
k) CipherPRNG based upon the construction described by Petit et al. [78], which offers protection against side channel attacks. There is a security proof that the scheme produces a sequence of random numbers indistinguishable from the uniform distribution.

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.

Short signatures, such as BLS and ZSS (BBS), are not as fast as EC, since this kind of constructions are based on bilinear pairings. Here, there is a tradeoff, because the signature can be roughly half the size of a regular ECDSA signature, but the verification algorithm must compute a bilinear pairing and, therefore, is less efficient. It is important to remark that the ability to compute bilinear pairings allows us to achieve many new cryptographic functionalities, such as identity based cryptography and certificateless encryption. Furthermore, the scheme ED25519 is a recently proposed digital signature cryptosystem that has been built over elligator curves [70], which offers advantages against side channel attacks and is a non-standard construction which may not be susceptible to surveillance manipulation.

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.

For instance, a recent improvement in mathematics [79][80] 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.

D. *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. For instance, Garbage Collections (GC) as well as optimizations and recompilations can be clearly identified in diagrams, as shown in Figure 5(A). The figure shows the time consumed by the first 300 executions of a pure-Java implementation of the AES algorithm, for both encryptions (E) and decryptions (D) of a small block of data, with a 128-bit key. The measurements were taken on a Samsung Galaxy S III (Quad-core 1.4 GHz Cortex-A9 processor, 1GB of RAM, and Android 4.1). The figure shows that at the very first moments of execution, the algorithm has 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, despite a few GC calls.

Due to the above mentioned limitations, two approaches of measurement have been used for the evaluation of cryptographic functions. The first one was the measurement of elapsed time for single cryptographic functions processing a single (small) 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.

Figure 5(B) exemplifies the first approach and shows the comparative performance of AES's encryptions (E) and decryptions (D) of a single block of data, for two cryptographic providers for Android: this CryptoLib (CSP), and BouncyCastle's [81] deployment for Android, SpongeCastle (SC) [82]. AES were setup to ECB mode and 128-bit key. The measurements were taken on a smartphone Samsung Galaxy S III (Quad-core 1.4 GHz Cortex-A9 processor, 1GB of RAM, and Android 4.1). The procedure consisted of processing a single block of data in a loop of 10,000 iterations.

In order to inhibit the negative influence of GC and JiTC, two metrics were taken: the average of all iterations and the 9th centile. None of them resulted in a perfect metric, but the 9th centile were able to reduce negative influence from GC and JiTC. For small data chunks, CSP is faster than SC.

The second approach for performance evaluation supposes that final users of mobile devices will not tuning their Java VMs with obscure configuration options in order

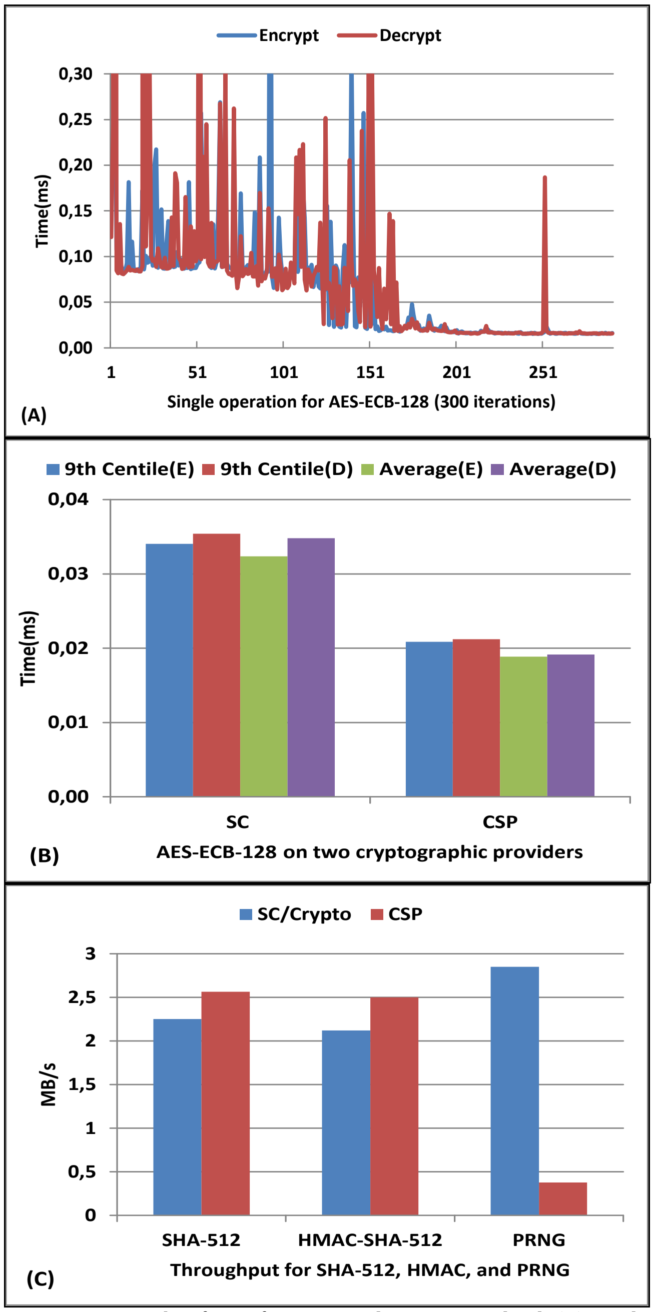


Figure 5. Approaches for performance evaluation on Android. (A) Single measurements suffer from GC and JiTC. (B) 9th centile and average show expected behavior, but are less accurate. (C) Throughput gives user's perceived responsiveness.

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 5(C) shows the throughput of SHA-256 and HMAC-SHA-256 implemented by CryptoLib (CSP) compared to SC. Also it shows the throughput for two instances of Pseudo Random Number Generator (PRNG): CSP's CipherPRNG compared to a SHA1PRNG available to Android apps through a provider called Crypto. 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 5 MB, in a loop of 500 iterations. It is interesting to observe that CSP and SC are quite similar in performance for SHA-256 and HMAC, CSP is slightly better. However, CSP's CipherPRNG has shown a low throughput, mostly because its construction is relatively inefficient, since it is based on block ciphers instead of hash functions. Nonetheless, this implementation is still a proof of concept and better timings are expected in the future.

Performance measurements for other implementations of non-standard cryptography were taken as well. 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 two smartphones: (i) LG Nexus 5 with processor 2.3 GHz quad-core Krait 400, 2GB of RAM, and 16GB of storage and (ii) Samsung Galaxy S III with processor of 1.4 GHz quad-core Cortex-A9, 1 GB

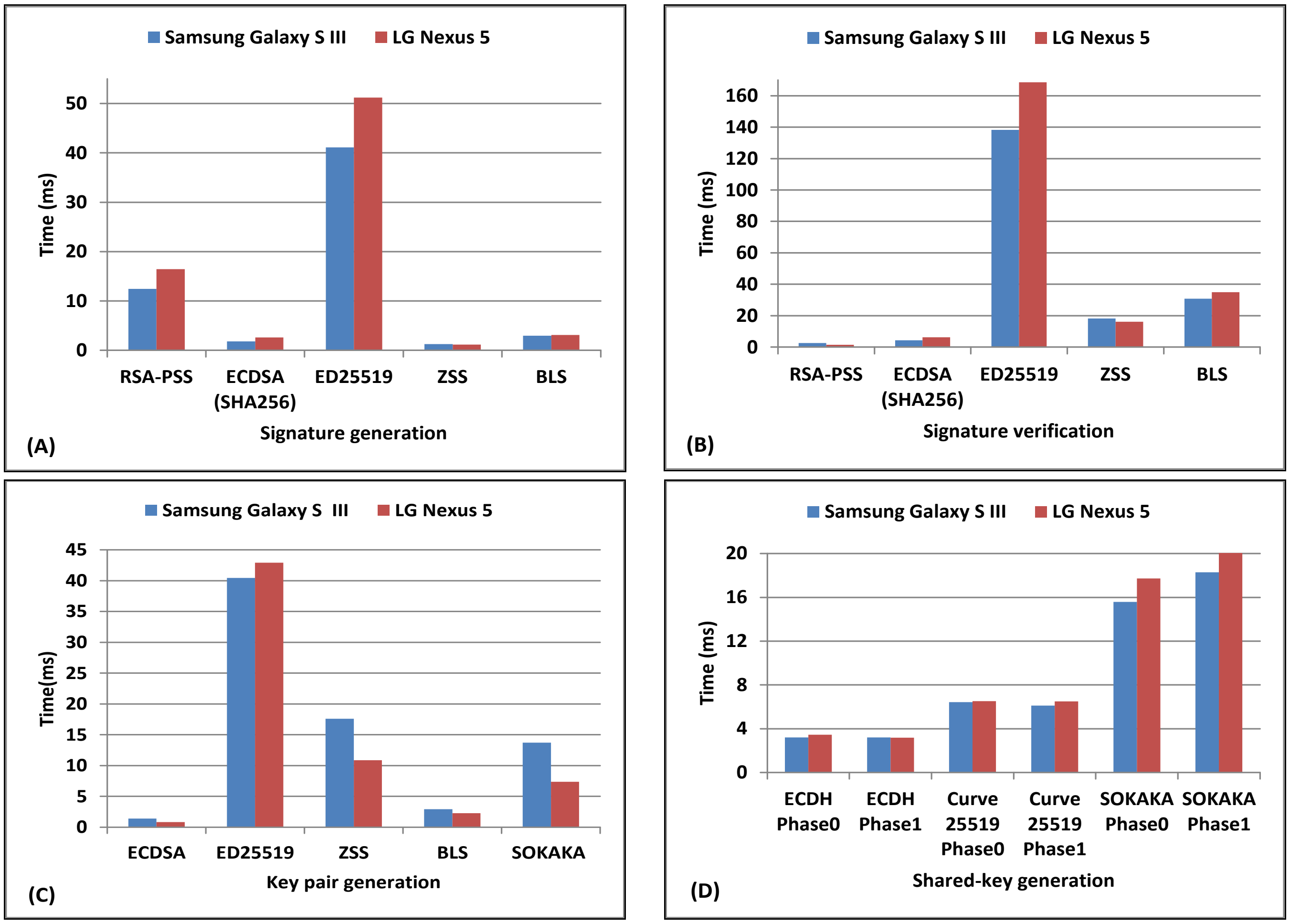


Figure 6. Performance evaluation of non-standard cryptography compared to standards. RSA uses 1024-bit key, all others have security level of 256-bit. Digital signatures: (A) generation, (B) verification, and (C) key pair generation. Key Agreement: (D) parameters generation and secret agreement.

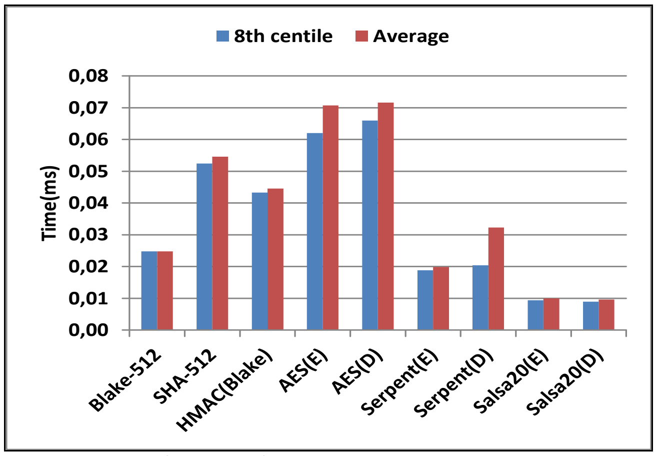


Figure 7. Performance of non-standard cryptography (symmetric encryption, secure hash, and MACs) compared to AES and SHA-512.

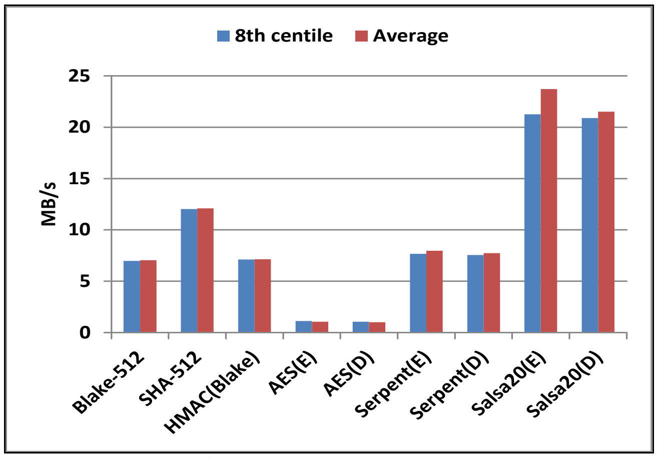


Figure 8. Throughput of non-standard cryptography (symmetric encryption, secure hash, and MACs) compared to AES and SHA-512.

of RAM, and 16 GB of storage.

Figure 6 shows two types of services: digital signatures at the top and key agreement (KA) at the bottom. The bar chart Figure 6(A) shows generation of digital signatures for five algorithms: RSA (1024-bit key), ECDSA (with SHA-256), ED25519, BLS and ZSS (BBS), all of them for 256-bit security. Traditionally, RSA is the slowest one. Elliptic curve cryptography, as in ECDSA, is faster. Short signatures, such as BLS and ZSS (BBS), are not as fast as EC. The scheme ED25519 is the slowest one. This implementation is still a proof of concept and better timings are expected after optimizations.

Bar chart of Figure 6(B) shows verification of digital signatures for five algorithms: RSA (1024-bit key), ECDSA (with SHA-256), ED25519, BLS and ZSS (BBS), with 256-bit security. Traditionally, RSA verification is the fastest one. Elliptic curve cryptography, as in ECDSA, is not that fast. Short signatures, such as BLS and ZSS (BBS), are terribly slow, due to complex arithmetic involved in bilinear pairings computations. ED25519 is the slowest one.

Figure 6(C) shows key pair generation for ED25519, BLS, ZSS (BBS) and SOKAKA, a pairings-based KA scheme, compared to ECDSA. Again, performance is slow for BLS, ZSS (BBS), and SOKAKA. ED25519 is the slowest. Figure 6(D) shows two KA schemes (Curve25519 and SOKAKA) compared to ECDH. ECDH is quite fast. Curve25519 is faster than SOKAKA.

Additional measurements were taken for symmetric, non-standard algorithms on the same Samsung Galaxy S III. Figure 7 shows time measurements of single-block operations for the following algorithms: (i) Blake 512 and HMAC with Blake compared to SHA-512; (ii) Serpent and Salsa20 compared to AES. Algorithms were setup with a 256-bit key, if needed. The bar chart shows both average and the 8^{th} centile of 10 thousand operations. Serpent is faster than homegrown AES, but Salsa20 is the fastest. Blake 512 is quite competitive to SHA-512 for small amounts of data.

Figure 8 tries to capture the perceived responsiveness and considers the throughput for the same symmetric algorithms, in megabytes per seconds (MB/s), to process a single file of 5MB, in a cycle of 500 iterations. The best throughput is given by Salsa20. Interestingly, Blake has shown slower performance than SHA-512 for large amounts of data.

VI. Encryptted FileSystem with Secure Deletion

In order to protect the secrecy of data during its entire lifetime, encrypted file systems must provide not only ways to securely store, but also reliably delete data, in such a way that recovering them from physical medium is almost impossible. The rationale behind the proposed solution is the actual possibility of performing secure deletion of files from ordinary Android applications, in user mode, without administrative privileges or operating system customization.

A. General description of the proposed solution

The proposed solution handles two cases according to the place where the deleted (or about to be deleted) file is stored:

1. The file is kept by the encrypted file system;
2. The file is logically deleted by the O.S.

1) Secure Deletion of Encrypted Files

The simplest way to fulfill the task of securely delete a file from an encrypted file system is to simply lose the encryption key of that file and then logically remove the file. This method does not need memory cleaning (purging) and is very fast. A prototype was built upon an Android port [44] for the EncFS encrypted file system [83]. Figure 9 illustrates the general behavior and functioning of the encrypted file system and its management application, called CryptoFM. The figure shows CryptoFM usage:

1. Inside CryptoFM, user sees a file system;
2. Inside, file names are decrypted on-the-fly;
3. Outside CryptoFM, user sees encrypted folders;
4. Inside, all file names are encrypted as well;
5. Outside, the file type is hidden;
6. Inside, corruptions are detected and monitored.

To accomplish the task of secure file deletion, the way EncFS manages cryptographic keys had to be modified. EncFS encrypts all files with a single master key derived from a password based encryption (PBE) function. It seems quite obvious that it is not feasible to change a master key and encrypt the whole file system every time a single file is deleted. On the other hand, if each file were encrypted with its own key, then that key could be easily thrown away, turning the deleted file irrecoverable.

The modification to EncFS consists of the following:

a) Use PBE to derive a master key MK;
b) Use a Key Derivation Function (KDF) to derive a File System Encryption Key (FSEK) from MK;
c) Use an ordinary key generation function (e.g., PRNG) to generate a File Encryption Key (FEK);
d) Encrypt files along with their names using FEK and encrypts FEK with FSEK and random IV;
e) Keep a mapping mechanism from FEK and IV to encrypted file (FEK||IV → file).

A simple way to keep that mapping is to have a table file stored in user space as application's data. Care must be taken to avoid accidentally or purposely remove that file when cleaning device's user space. In Android devices, this can be done by rewriting the default activity responsible for deleting application's data. An application-specific delete activity would provide a selective deletion of application's data or deny any deletion at all. The removal from table of the FEK and IV makes a file irrecoverable. The ordinary delete operation then return storage space of that file to operating system. Figure 10 depicts the solution.

Another way to keep track of keys and files is to store the pair {FEK, IV} inside the encrypted name of the encrypted file. In this situation, a file has to be renamed before removed from the encrypted file system. The rename operation destroys the FEK and makes file irrecoverable. The ordinary delete operation then return storage space to operating system.

It is interesting to note that the proposed solution contributes to solve some known security issues of EncFS [84][85]. By using distinct keys for every file, a Chosen Ciphertext Attack (CCA) against the master key is inhibited. Also, it reduces the impact of IV reuse across encrypted files. Finally, it eliminates the watermarking vulnerability, because a single file imported twice to EncFS will be encrypted with two distinct keys and IVs.

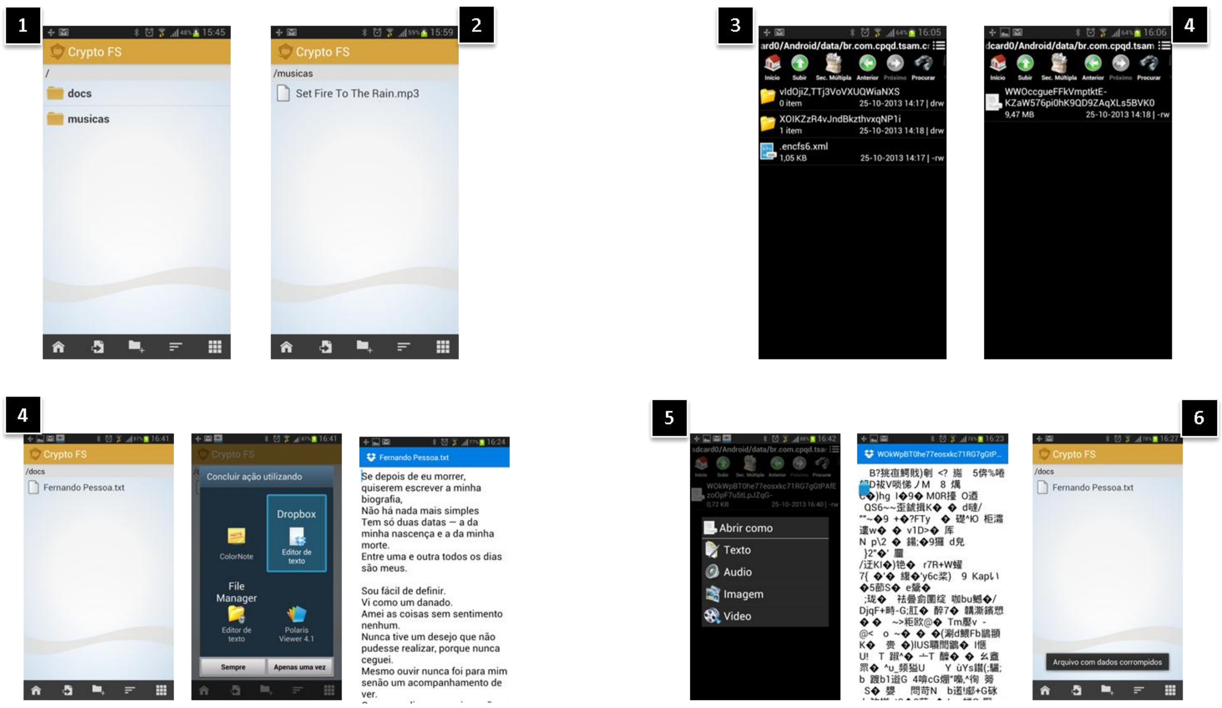
Finally, the key derivation function is based upon PBKDF2 standard [59], keys and IVs are both 256 bits, and the table for mapping the pair {key, IVs} to files is kept by an SQLite scheme accessible only by the application.

2) Secure deletion of ordinary files

In this context, a bunch of files were logically deleted by the operating system for the benefit of the user, but they left sensitive garbage in the memory. Traditional solutions for purging memory cells occupied by those files are innocuous, because there is no way to know, from user's point of view, where purging data will be written.

An instance of this situation occurs when a temporary file is left behind by an application and manually deleted. This temporary file may be a decrypted copy of an encrypted file kept by the encrypted file system. Temporary unencrypted copies of files are necessary in order to allow other applications to handle specific file types, e.g., images, documents, and spreadsheets.

Whether temporary files will or will not be imported back to the encrypted file system, they have to be securely removed anyway. A premise is that the files to be removed are not in use by any application. The secure deletion occurs


1
2
3
4
4
5
6
Crypto FS
/
docs
musicas
/musicas
Set Fire To The Rain.mp3
ard0/Android/data/br.com.cpqd.tsam.cr
Início
Subir
Sec. Múltipla
Anterior
Próximo
Procurar
vldOjiZ,TTj3VoVXUQWiaNXS
0 item 25-10-2013 14:17 | drw
XOIKZzR4vJndBkzthvxqNP1i
1 item 25-10-2013 14:18 | drw
.encfs6.xml
1,05 KB 25-10-2013 14:17 | -rw
dcard0/Android/data/br.com.cpqd.tsam
WWOccgueFFkVmptktE-KZaW576pi0hK9QD9ZAqXLs5BVK0
9,47 MB 25-10-2013 14:18 | -rw
/docs
Fernando Pessoa.txt
Concluir ação utilizando
ColorNote
Dropbox
Editor de texto
File Manager
Editor de texto
Polaris Viewer 4.1
Sempre
Apenas uma vez
Fernando Pessoa.txt
sdcard0/Android/data/br.com.cpqd.tsa
WOkWpBT0he77eosxkc71RG7gGtPAfE zoOpF7u5tLpJZqG-
0,72 KB 25-10-2013 16:40 | -rw
Abrir como
Texto
Audio
Imagem
Video
WOkWpBT0he77eosxkc71RG7gGtP...
Arquivo com dados corrompidos

Se depois de eu morrer,
quiserem escrever a minha
biografia,
Não há nada mais simples
Tem só duas datas — a da
minha nascença e a da minha
morte.
Entre uma e outra todos os dias
são meus.

Sou fácil de definir.
Vi como um danado.
Amei as coisas sem sentimento
nenhum.
Nunca tive um desejo que não
pudesse realizar, porque nunca
ceguei.
Mesmo ouvir nunca foi para mim
senão um acompanhamento de
ver.

Figure 9. General behavior and functioning of the encrypted file system and its management application CryptoFM.

in three steps:

1) Logically remove targeted files with ordinary deletion;
2) Write a temporary file of randomized content that occupies all storage's free space;
3) When there is no free space anymore, logically delete that random file. That action purges all free storage in a way that no sensitive data is left behind.

The final result of this procedure is a flash storage free of sensitive garbage. Steps two and three can be encapsulated as a single function, called memory purging, and performed by an autonomous application. That application would be activated by the user whenever she needs to clean storage from sensitive garbage. The proposed solution adopted variations of this behavior.

Unfortunately, this procedure has two drawbacks. First, it takes time proportional to the size of the free space to be cleaned and the speed of memory writes. Second, this procedure, in the long term, if used with high frequency, has the potential to shorten the lifetime of flash memories.

In order to minimize the negative impact over memory life and avoid excessive delays during operation, steps two and three from above should not be carried out for every single file deleted from the system.

3) Limitations of the solution

The protection of cryptographic keys is of major importance. In spite of being stored encrypted, decrypted just before being used, and then released, the protection of cryptographic keys relies on Android security and the application confinement provided by that operating system. The proposed solution for memory purging is supposed to work in user-mode, as an ordinary mobile app, without administrative access, with no need for operating system modification, and using off-the-shelf devices. These decisions have consequences for security.

First of all, the solution is highly dependent on the way flash-based file systems and controllers behave. Briefly speaking, when the flash storage is updated, the file system writes a new copy of the changed data to a fresh memory block, remaps file pointers, and then erases the old memory blocks, if possible, but not certainly. This constrained design actually enables alternative implementations discussed further.

A second issue is that the solution is not specifically concerned about the type of physical memory (e.g., internal, external SD, NAND, and NOR) as long as it behaves like a flash-based file system. The consequence is that only software-based attacks are considered and physical attacks are out of scope.

Additionally, the use of random files is not supposed to have any effect on the purging assurance, but provides a kind of low-cost camouflage for cryptographic material (e.g., keys or parameters) accidentally stored on persistent media. An entropy analysis would not be able to easily distinguish specific random data as potential security material, because huge amounts of space would look random. Of course, this software-based camouflage cannot be the only way to prevent such attacks, but it adds to a defense in depth approach to security at almost no cost.

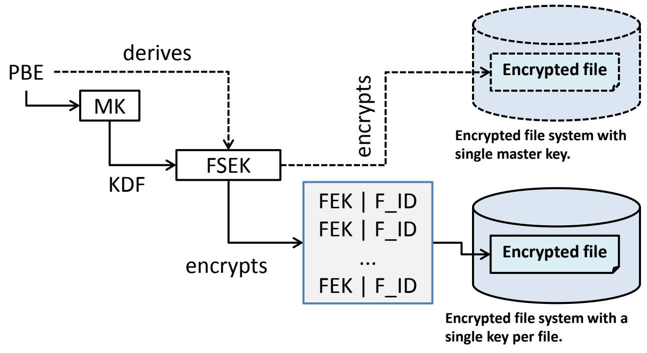


Figure 10. Extending an encrypted file system for secure deletion.

Finally, the purging technology described has passed all recovery tests performed with publicly available recovery tools, such as PhotoRec [86] and similar. That means, after purging, none of the recovery tools were able to recovery any deleted file. This confirms the feasibility of the purging technology for final users. On the other hand, advanced users may need deeper security assessments over physical hardware in order to trust the actual extend of the security provided by the proposed solution.

B. *Alternative implementaions*

The proposed solution for memory purging is a general policy for purging flash memories, and can be implemented in various ways, ranging from simple to complex implementations. In fact, a general solution has to offer different trade-offs among security requirements, memory life, and system responsiveness. The authors have identified three points for customization:

1. The period of execution for the purging procedure;
2. The size and quantity of random files;
3. The frequency of files creation/deletion.

Different trade-offs among the three customization points previously identified were implemented and evaluated. In all of them, the random file created in order to clean storage free space is called bubble, after the metaphor of soap cleaning bubbles over a dirty surface. These alternatives are discussed in next paragraphs.

1) Static single bubble

The simplest solution described in this text implements the idea of a single static bubble that increases in size until it reaches the limit of free space, and then bursts. This solution is adequate for the cases when storage has to be cleaned in the shortest period of time, with no interruption. A disadvantage is that other concurrent application can starve out of storage.

This solution is adequate when nothing else is happening, but the purging. Figure 11 illustrates, in four simple steps, the general behavior of this implementation:

1. Sensitive files are deleted logically;
2. The purging bubble is created and grows to occupy all available storage;
3. The bubble is logically removed when it reaches the limit of available storage;

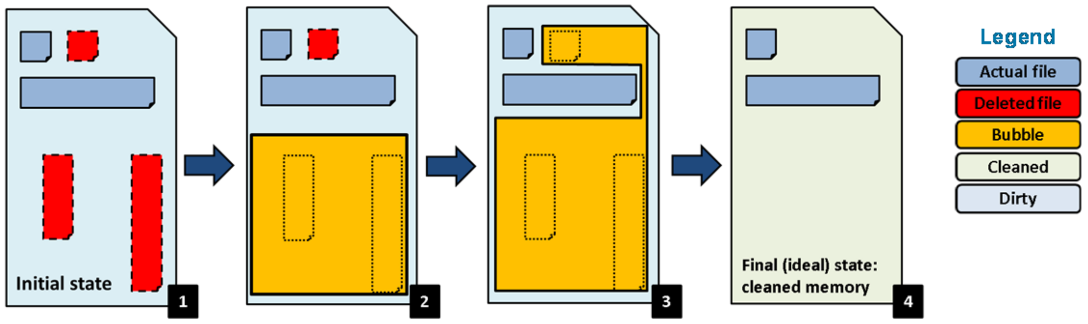


Figure 11. Purging strategy #1: Static Single Bubble.

4. The removed bubble leaves its waste, which overwrites any sensitive waste previously left in storage.

In that figure, an actual file is shown in blue, logically deleted files are in red, the bubble are in orange, dirty memory is light blue and purged memory is light grey.

2) Moving or sliding (single) bubble

In this alternative, a single bubble periodically moves itself or slides from one place to another. The moving bubble has size of a fraction of free space. For example, if bubble size is n^{-1} of free space, the moving bubble covers all free storage after n moves, considering the amount of free space does not change. A move is simply the rewriting of the bubble file, since flash memories will perform a rewrite in a different place. Figure 12 illustrates, in five simple steps, the general behavior of this implementation:

1. Sensitive files are deleted logically;
2. The purging bubble is created with a fraction of the available storage;
3. The purging bubble moves due to rewriting behavior;
4. The bubble is logically removed when it has covered all the free space and have reached the limit;
5. The removed bubble leaves its waste, which overwrites any sensitive waste previously left in storage.

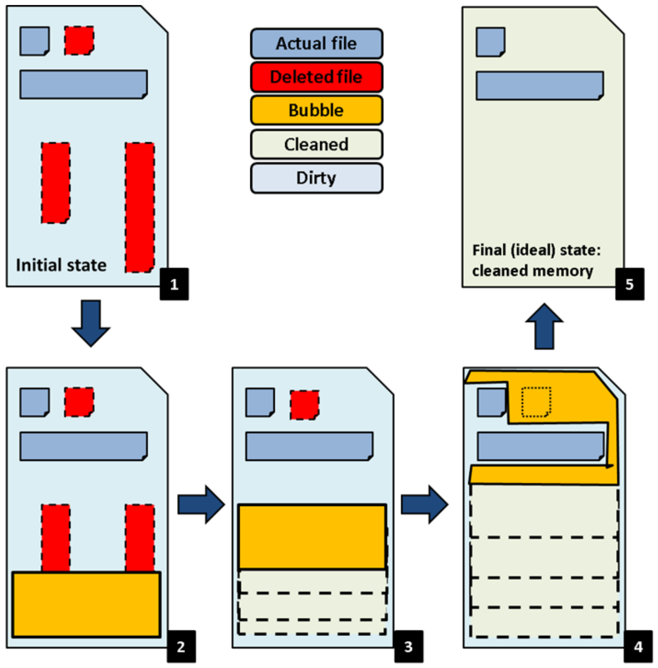


Figure 12. Purging strategy #2: Sliding single bubble.

In a period of time equals to (T*(n/2)), where T is the time between moves, the chance of finding sensitive garbage in memory is 50%. This solution is adequate when storage has a low to moderate usage by concurrent applications. This solution preserves system responsiveness (usability) but diminishes security.

3) Moving or sliding (multiple) bubbles

This alternative uses more than one bubble instead of a single one. The size and amount of bubbles are fixed. For instance, if bubble size is n^{-1} of free space, two moving bubble covers all free storage space after n/2 moves each. The advantage of this method is to potentially accelerate memory coverage, reducing opportunity for memory compromising. Figure 13 illustrates the general behavior of this implementation:

1. Sensitive files are deleted logically;
2. The purging bubbles are created with a fraction of the available storage;
3. The bubbles move due to rewriting behavior;
4. Removed bubbles leave their wastes, which overwrite any sensitive waste previously left in storage.

In the example, two bubbles of size 1/n each can move at every T/2 period, and then concluding in (T*n). Alternatively, they can move at period T and terminate in 2*T*n, and so on. This solution is adequate when storage has a moderate usage by concurrent applications. This solution is probabilistic in the sense that as smaller the duration of T and greater the size of bubbles, greater the chance of successfully clean all memory.

4) Sparkling bubbles

This solution varies the size and amount of bubbles. The idea is to create a bunch of mini bubbles that are sparkled over free storage space. Bubbles are created and instantly removed at period T, which can be constant or random between zero and T. The sparking of bubbles stops when the sum of sizes for all created bubbles surpasses free space.

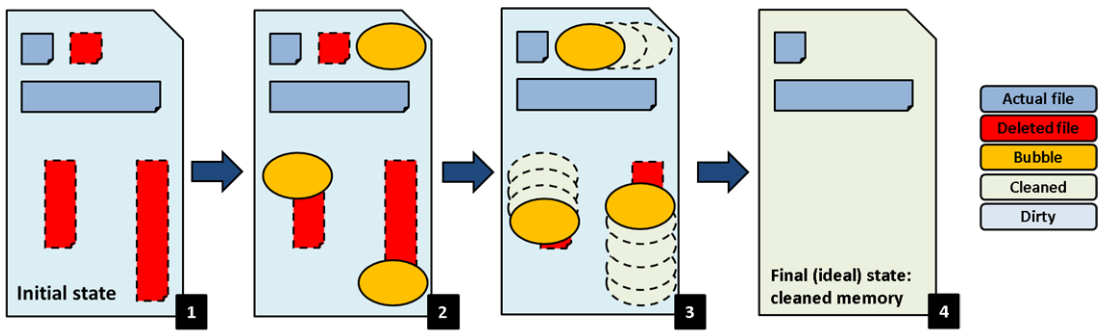


Figure 13. Purging strategy #3: Sliding (multiple) bubbles.

Bubble size can be small enough to not affect other applications. Figure 14 illustrates the general behavior of this implementation:

1. Sensitive files are deleted logically;
2. The bubbles are created with random size between a (specified) minimum and maximum;
3. The bubbles are removed and recreated concurrently;
4. The bubbles stop being created when the sum of their sizes reaches the size of free space;
5. Removed bubbles leave their wastes, which overwrite any sensitive waste previously left in storage.

This solution is adequate when storage has a moderate to high usage by concurrent applications. This solution is probabilistic in the sense that as smaller the duration of T, greater the chance of successfully clean the whole memory.

C. *Performance evaluation*

The four alternative implementations were compared according to their throughput for memory cleaning. That means, the rate at which data are purged, in gigabytes per minute (GB/min). This measure of purging speed tends to be more useful to compare storages of different size, such as internal and external memory. Performance tests were performed in two smartphones of type Motorola Atrix MB860, with Android 2.3.6 operating system, dual core 1GHz processor, 1GB of RAM and 16GB of internal storage (only 11 GB available to the end user). It was also used an SD Card (Class C) of 2GB. Random files created for purging had size of at most 2 GB or one tenth of free space. Performance measures were carried out in three scenarios:

- Scenario 1: mostly empty storage (~ 0-19%);
- Scenario 2: partially occupied storage (~ 20-80%);
- Scenario 3: mostly occupied storage (~ 81-99%).

In each scenario, both the internal and the external storage (SD card) were covered. Performance comparisons are structured as follows. First, a comparison is made between purging strategies for each occupancy scenario. Then, comparison is made between different occupancy scenarios for a specific strategy.

The implementations of the four purging strategies used concurrent threads if needed. The implementations of single static bubble and single sliding bubble used a single thread. The implementation of multiple sliding bubbles used two threads. The implementation of the mini-random bubbles used a minimum of five threads and at most twenty threads.

1) Scenario 1 – mostly empty storage

Performance measures for this scenario are shown in Figure 15(A). The storages were empty (0% occupancy). For this scenario, the following observations can be made:

a) The second purging strategy (single sliding bubble) is the fastest one. Apparently, this is because rewrite a single smaller file size is more efficient than continuously increase the size of a huge file;
b) The strategies with multiple bubbles are slower than the strategies with a single bubble. Probably, this is due to the overhead of managing multiple threads.
c) The higher the number of threads, worse the overall performance of purging, in slower CPUs. However, multi-bubble strategies are not blocking;
d) Purging of SD card was consistently slower in all cases.

Finally, data suggest that, when there is no competition for storage and storage is almost empty, sliding single bubble is the strategy that offers the best throughput. In situations

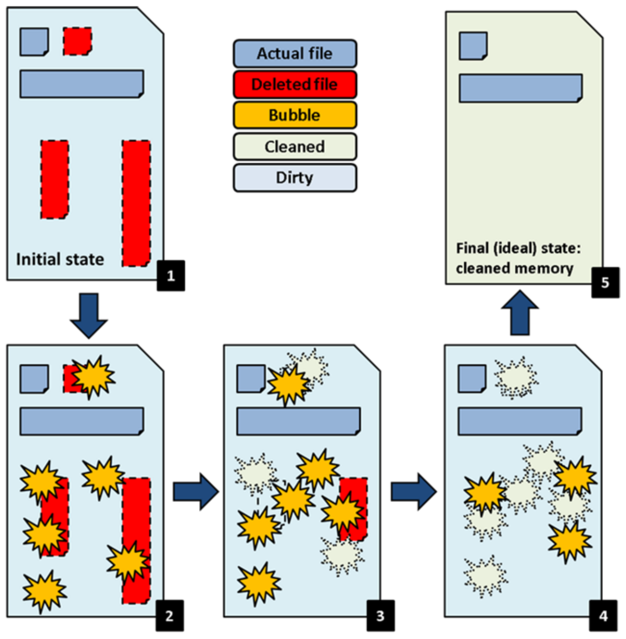


Figure 14. Purging strategy #4: Sparking bubbles.

where there is high competition for access to internal storage, multiple sliding bubbles seem to be more appropriate than the mini random bubbles.

2) Scenario 2 – partially occupied storage

Performance measures for this scenario are shown in Figure 15(B). The storages were partially occupied (30% occupancy). The following observations can be made:

a) The second strategy (single sliding bubble) is still the fastest one;

b) The strategies with multiple bubbles are slower than the strategies with a single bubble;

c) The higher the thread count, the worse the overall performance of purging;

d) Purging of SD card was consistently slower in all cases.

Finally, data suggest that, when there is no competition for storage and it is partially occupied, the single sliding bubble is still the strategy that offers the best throughput. However, single static bubble is very competitive. In situations where the competition for the internal storage is high, the throughputs for multiple sliding bubbles and mini random bubbles are quite similar.

3) Scenario 3 – mostly occupied storage

Performance measurements for this scenario are shown in Figure 15(C). The storages were nearly full (94% occupancy). The following observations can be made:

a) The first strategy (single static bubble) is just slightly faster than the second one (single sliding bubble);

b) The throughput of the strategies with multiple bubbles is close to the throughput of strategies with a single bubble, but showing a slightly worse performance;

c) The overall performance is still slightly worse with the increase number of threads;

d) Purging of SD card was consistently slower.

Data suggest that, when there is no competition for storage and its occupation is close to full capacity, the single bubble strategies (static or sliding) offer the best throughput.

4) Comparisom among strategies

Figure 16 compares all four strategies in different scenarios of occupancy (0%, 30% and 94%). All amounts are in GB/min. The following observations can be made:

1. In Figure 16(A), throughput of the single static bubble strategy improves with increasing storage occupation;
2. In Figure 16(B), throughput of the single sliding bubble gets worse with increased storage occupancy. This may be due to the slower treatment of memory rewriting;
3. In Figure 16(C), throughput of multiple sliding bubbles improves with increasing storage occupation. The use of two threads compensates for the relative slowness of the bubble rewriting;
4. In Figure 16(D), throughput of multiple random bubbles improves with increasing storage occupation.

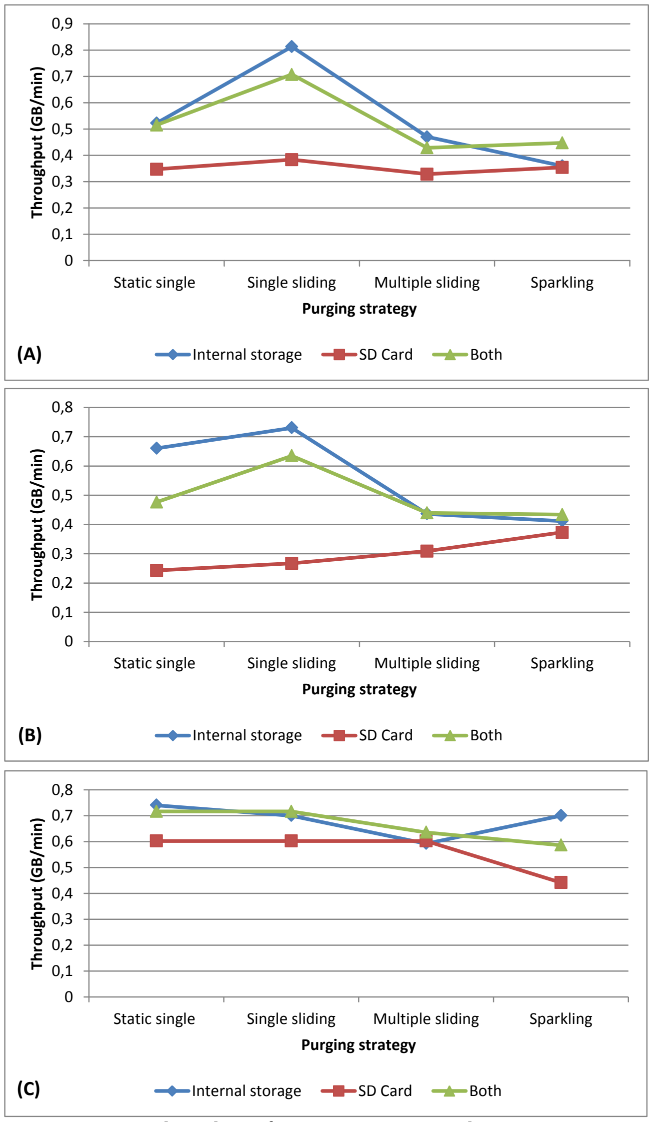


Figure 15. Throughputs for purging strategy in three scenarios.

The use of multiple threads is combined with rapid generation of small bubbles.

The measurements show that the purging strategy with single sliding bubble has the highest throughput in average, being considered most appropriate in general. However, the static bubble is very competitive, though. In situations where there is high competition for internal storage, the throughput of strategies with multiple bubbles (sliding and random) is similar.

VII. INTEGRATED VIEW: PUTTING IT ALL TOGETHER

Among all technical challenges concerning the development of security features for applications on modern, Android-based, mobile devices, one of major importance is the integration of all these features into a security-ware framework. Figure 17 illustrates the high-level architecture of the proposed framework, where an application container encapsulates all security features, including cryptography, key management, contact management, secure storage and deletion, access control, and mediated access to server-side applications. All these features are accessible to applications by means of APIs and services. Also, the framework promotes integration among mobile applications. For instance, the encrypted file system can be accessed by trusted applications inside the container.

Two main objectives drove the proposed architecture shown in Figure 17. The first one was to build a family of secure communication services over data packets (or over IP), through smartphones on public networks (e.g., 3G, 4G, Wi-Fi). The second was to develop tools for integrity checking and remote monitoring of smartphones, as well as techniques for active investigation on mobile platforms.

At the back office, the framework is supported by a laboratory for mobile security, which is able to carry out assessments on mobile environments, including platforms, applications and communications, as well as security analysis of mobile malware. The knowledge acquired by the lab team feeds the development team with security controls and counter measures. A private cloud provides services to the development team. Not only security services are provided, but also hosting for server-side applications.

VIII. CONCLUDING REMARKS

This paper discussed design and implementation issues on the construction of an integrated framework for securing both communication and storage of sensitive information over Android devices.

This text has shown how cryptographic services can be crafted to adequately fit secure communication services as well secure storage and deletion mechanisms, in such a way that security is kept transparent to the user, without being sacrificed. Also, a well-defined architecture allowed the selection and use of non-standard cryptography on a cryptographic library for Android.

The cryptographic library actually consists of both standard and non-standard cryptographic algorithms. Performance measurements were taken in order to compare cryptographic providers. Despite all difficulties to obtain realistic data, experiments have shown that standard cryptography can be competitive to other implementations. On the other hand, non-standard cryptography has shown low performance that can possibly limit its use in real time applications. However, their value consists in offering secure alternatives to possibly compromised standards. In fact,

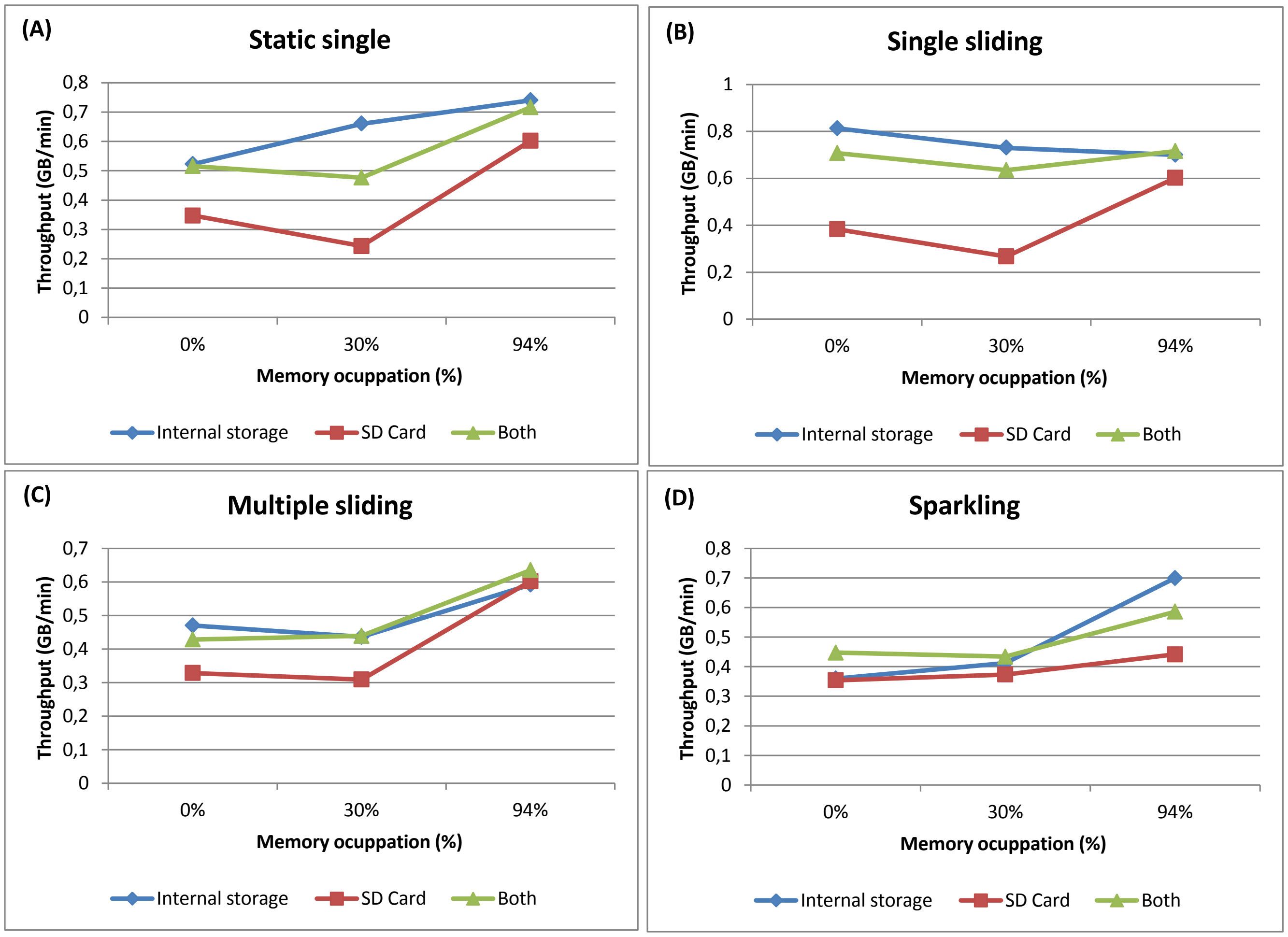


Figure 16. Throughputs by memory occupancy for purging strategy.

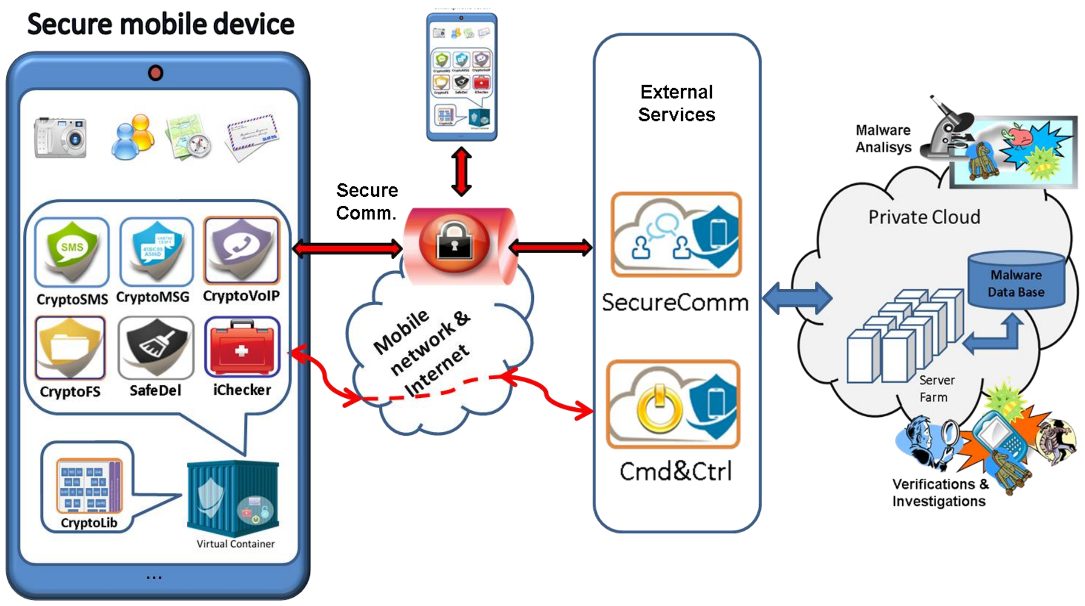


Figure 17. Framework high-level architecture with secure communication, trust management and back office services.

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.

Finally, the paper discussed the implementation of two user-level approaches to perform secure deletion of files. One works on secure deletion of encrypted files and the other handles deletion assurance of ordinary (unencrypted) files. Secure deletion of encrypted files was fully integrated to an encrypted file system and is transparent to the user. Secure deletion of ordinary files was fulfilled by an autonomous application activated under the discretion of the user. Performance measurements have shown that the approach is feasible and offers interesting trade-offs between time and deletion assurance.

In the short term, future work comprises the inclusion of additional secure applications to the mobile security framework, such as SMS, email, voice mail and VoIP. In the long run, the framework should evolve to a mobile platform for remote monitoring and fine-grained control of secure devices. Finally, as secure computing platforms become common place in mobile devices, the framework should be integrated to such features and provide strong, hardware-based protection to cryptographic keys.

Acknowledgment

The authors would like to acknowledge professor Julio Lopez, from University of Campinas, for his helpful comments. 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. M. Braga and A. H. G. Colito, “Adding Secure Deletion to an Encrypted File System on Android Smartphones,” The Eighth International Conference on Emerging Security Information, Systems and Technologies (SECURWARE), 2014, pp. 106-110.

[2] A. M. Braga and D. C. Schwab, “Design Issues in the Construction of a Cryptographically Secure Instant Message Service for Android Smartphones,” The Eighth International Conference on Emerging Security Information, Systems and Technologies (SECURWARE), 2014, pp. 7-13.

[3] A. M. Braga and E. M. Morais, “Implementation Issues in the Construction of Standard and Non-Standard Cryptography on Android Devices,” The Eighth International Conference on Emerging Security Information, Systems and Technologies (SECURWARE), 2014, pp. 144-150.

[4] A. M. Braga, “Integrated Technologies for Communication Security on Mobile Devices,” The Third International Conference on Mobile Services, Resources, and Users (Mobility), 2013, pp. 47–51.

[5] A. M. Braga, E. N. Nascimento, and L. R. Palma, “Presenting the Brazilian Project TSAM – Security Technologies for Mobile Environments,” Proceeding of the 4th International Conference in Security and Privacy in Mobile Information and Communication Systems (MobiSec 2012). LNICST, vol. 107, 2012, pp. 53-54.

[6] 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.

[7] J. Menn, “Experts report potential software ‘back doors’ in U.S. standards,” retrived [May 2015] from http://www.reuters.com/article /2014/07/15/usa-nsa-software-idUSL2N0PP2BM20140715?irpc=932.

[8] NIST Removes Cryptography Algorithm from Random Number Generator Recommendations. Retrieved [May 2015] from http://www.nist.gov/itl/csd/sp800-90-042114.cfm.

[9] Java Cryptography Architecture (JCA) Reference Guide. Retrieved [May 2015] from docs.oracle.com/javase/7/docs/technotes/guides/ security/crypto/CryptoSpec.html.

[10] Java Cryptography Extension Unlimited Strength Jurisdiction Policy Files 7 Download. Retrieved [May 2015] from www.oracle.com/ technetwork/java/javase/downloads/jce-7-download-432124.html.

[11] Java Cryptography Architecture (JCA) Reference Guide. Retrieved [May 2015] from docs.oracle.com/javase/7/docs/technotes/guides/ security/crypto/CryptoSpec.html.

[12] The Java HotSpot Performance Engine Architecture. Retrived [May 2015] from www.oracle.com/technetwork/java/whitepaper-135217.html.

[13] Tuning Garbage Collection with the 5.0 Java Virtual Machine. Retrived [May 2015] from http://www.oracle.com/technetwork/ java/gc-tuning-5-138395.html.

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

[15] A. Lux and A. Starostin, “A tool for static detection of timing channels in Java,” Journal of Cryptographic Engineering, vol. 1, no. 4, Oct. 2011, pp. 303–313.

[16] D. Bornstain, “Dalvik VM Internals,” retrieved [May 2015] from sites.google.com/ site/io/dalvik-vm-internals.

[17] H. Oh, B. Kim, 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.

[18] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, “A study of Android application security,” in Proceedings of the 20th USENIX conference on Security (SEC’11), USENIX Association, Berkeley, CA, USA, 2011, p. 21.

[19] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, “An empirical study of cryptographic misuse in android applications,” in Proceedings of the 2013 ACM SIGSAC conference on Computer and Communications Security (CCS ’13), 2013, pp. 73–84.

[20] P. Gutmann, “Lessons Learned in Implementing and Deploying Crypto Software,” Usenix Security Symposium, 2002.

[21] NIST SP 800-38A. Recommendation for Block Cipher Modes of Operation. 2001. Retrieved [May 2015] from csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf.

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

[23] P. Gutmann, “Secure deletion of data from magnetic and solid-state memory,” proceedings of the Sixth USENIX Security Symposium, San Jose, CA, vol. 14, 1996.

[24] NSA (2012). Enterprise Mobility Architecture for Secure Voice over Internet Protocol. Mobility Capability Package - Secure VoIP, V 1.2.

[25] A. Voyiatzis, K. G. Stefanidis, and D. N. Serpanos, “Increasing lifetime of cryptographic keys on smartphone platforms with the controlled randomness protocol,” in Proceeding of the Workshop on Embedded Systems Security (WESS’11), New York, NY, USA, 2011.

[26] J. Grosschadl and D. Page, “Efficient Java Implementation of Elliptic Curve Cryptography for J2ME-Enabled Mobile Devices,” Cryptology ePrint Archive, Report Nr. 2011/712, 2011.

[27] M. Smith, C. Schridde, B. Agel, and B. Freisleben, “Secure mobile communication via identity-based cryptography and server-aided computations,” J. Supercomput, vol. 55, no. 2, Feb. 2011, pp. 284-306.

[28] A. De Caro and V. Iovino, “ jPBC: Java pairing based cryptography,” In Proceedings of the IEEE Symposium on Computers and Communications (ISCC '11). IEEE Computer Society, 2011.

[29] ENISA, “Algorithms, key size and parameters report,” nov. 2014. Retrived [May 2015] from www.enisa.europa.eu/activities/identity-and-trust/library/deliverables/algorithms-key-size-and-parameters-report-2014.

[30] ENISA. “Study on cryptographic protocols,” nov. 2014. Retrived [May 2015] from https://www.enisa.europa.eu/activities/identity-and-trust/library/deliverables/study-on-cryptographic-protocols.

[31] B. Xuefu and Y. Ming, “Design and Implementation of Web Instant Message System Based on XMPP,” Proc. 3rd International Conference on Software Engineering and Service Science (ICSESS), Jun. 2012, pp. 83-88.

[32] D. T. Massandy and I. R. Munir, “Secured Video Streaming Development on Smartphones with Android Platform,” Proc. 7th International Conference on Telecommunication Systems, Services, and Applications (TSSA), Oct. 2012, pp. 339-344.

[33] S. Schrittwieser et al., “Guess Who's Texting You? Evaluating the Security of Smartphone Messaging Applications,” in Proc. 19th Network & Distributed System Security Symposium, Feb. 2012.

[34] Off-the-Record Messaging webpage. Retrieved [May 2015] from otr.cypherpunks.ca.

[35] H. Krawczyk, “SIGMA: “The ‘SIGn-and-MAc’ approach to authenticated Diffie-Hellman and its use in the IKE protocols,” Advances in Cryptology-CRYPTO 2003, Springer Berlin Heidelberg, 2003, pp. 400-425.

[36] W. Diffie and M. Hellman, “New Directions in Cryptography,” IEEE Transact. on Inform. Theory, vol. 22, no. 6, Nov. 1976, pp. 644-654.

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

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

[39] Piercing Through WhatsApp’s Encryption. Retrieved [May 2015] from blog.thijsalkema.de/blog/2013/10/08/piercing-through-whatsapp-s-encryption.

[40] A. Greenberg, “Whatsapp just switched on end-to-end encryption for hundreds of millions of users,” Retrieved [May 2015] from www.wired.com/2014/11/whatsapp-encrypted-messaging.

[41] D. Boneh and R. J. Lipton, “A Revocable Backup System,” in USENIX Security, 1996, pp. 91-96.

[42] K. Sun, J. Choi, D. Lee, and S.H. Noh, “Models and Design of an Adaptive Hybrid Scheme for Secure Deletion of Data in Consumer Electronics,” IEEE Transactions on Consumer Electronics, vol. 54, no. 1, Feb. 2008, pp.100-104.

[43] S. M. Diesburg and A. I. A. Wang, “A survey of confidential data storage and deletion methods,” ACM Computing Surveys (CSUR), vol. 43, no.1, p.2, 2010.

[44] Z. Wang, R. Murmuria, and A. Stavrou, “Implementing and optimizing an encryption filesystem on android,” in IEEE 13th International Conference on Mobile Data Management (MDM), 2012, pp. 52-62.

[45] J. Reardon, S. Capkun, and D. Basin, “Data node encrypted file system: Efficient secure deletion for flash memory,” in USENIX Security Symposium, 2012, pp. 333-348.

[46] J. Reardon, C. Marforio, S. Capkun, and D. Basin, “User-level secure deletion on log-structured file systems,” in Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, 2012, pp. 63-64.

[47] J. Reardon, D. Basin, and S. Capkun, “On Secure Data Deletion,” Security & Privacy, IEEE , vol. 12, no. 3, May-June 2014, pp.37-44.

[48] J. Reardon, D. Basin, and S. Capkun, “Sok: Secure data deletion,” in IEEE Symposium on Security and Privacy, 2013, pp. 301-315.

[49] J. Reardon, H. Ritzdorf, D. Basin, and S. Capkun, “Secure data deletion from persistent media,” in Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (CCS '13). ACM, New York, NY, USA, 2013, pp. 271-284.

[50] A. Skillen and M. Mannan, “On Implementing Deniable Storage Encryption for Mobile Devices,” in 20th Annual Network & Distributed System Security Symposium, February 2013, pp. 24-27.

[51] A. Skillen and M. Mannan, “Mobiflage: Deniable Storage Encryption for Mobile Devices,” IEEE Transactions on Dependable and Secure Computing, vol. 11, no. 3, May-June 2014, pp.224-237.

[52] P. Saint-Andre, K. Smith, and R. Tronçon, "XMPP: The Definitive Guide - Building Real-Time Applications with Jabber Technologies," O'reilly, 2009.

[53] W. Mao, "Modern cryptography: theory and practice", PTR, 2004.

[54] EJBCA PKI CA. Retrieved [May 2015] from http://www.ejbca.org.

[55] S. Fahl, M. Harbach, and H. Perl, "Rethinking SSL development in an appified world," in Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security - CCS '13 (2013), 2013, pp. 49–60.

[56] Java Cryptography Architecture Oracle Providers Documentation for Java Platform Standard Edition 7. Retrieved [May 2015] from docs.oracle.com/javase/7/docs/technotes/guides/security/SunProviders.html.

[57] Java Cryptography Architecture Standard Algorithm Name Documentation for Java Platform Standard Edition 7. Retrieved [May 2015] from docs.oracle.com/javase/7/docs/technotes/guides/security/StandardNames.html.

[58] How to Implement a Provider in the Java Cryptography Architecture. Retrieved [May 2015] from docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/HowToImplAProvider.html.

[59] B. Kaliski, "PKCS #5: Password-Based Cryptography Specification," Version 2.0, RFC 2898. Retrieved [May 2015] from tools.ietf.org/html/rfc2898.

[60] J. Bos, D. Osvik, and D. Stefan, "Fast Implementations of AES on Various Platforms," 2009. Retrieved [May 2015] from eprint.iacr.org/2009/501.pdf.

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

[62] T. St. Denis. "Cryptography for Developers," Syngress, 2007.

[63] P. Barreto, AES Public Domain Implementation in Java. Retrieved [May 2015] from www.larc.usp.br/~pbarreto/JAES.zip.

[64] NIST FIPS-PUB-186. Digital Signature Standard (DSS). Retrieved [May 2015] from csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf.

[65] NIST FIPS-PUB-180-4. Secure Hash Standard (SHS). March 2012. Retrieved [May 2015] from csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf.

[66] NIST FIPS-PUB-198. The Keyed-Hash Message Authentication Code (HMAC). Retrieved [May 2015] from csrc.nist.gov/publications/fips/fips198/fips-198a.pdf.

[67] D. Aranha and C. Gouvêa, RELIC Toolkit. Retrieved [May 2015] from code.google.com/p/relic-toolkit.

[68] D. J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange, "Elligator: elliptic-curve points indistinguishable from uniform random strings," in Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, CCS '13, New York, NY, USA, 2013, pp. 967–980.

[69] D. Hankerson, A. J. Menezes, and S. Vanstone. "Guide to Elliptic Curve Cryptography," Springer-Verlag, New York, Inc., Secaucus, NJ, USA, 2003.

[70] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and Bo-YinYang, "High-speed high-security signatures," Journal of Cryptographic Engineering, vol. 2, no. 2, pp. 77–89, 2012.

[71] NIST FIPS PUB 186-2. Digital Signature Standard (DSS). Retrieved [May 2015] from csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf.

[72] R. Sakai, K. Ohgishi, and M. Kasahara. "Cryptosystems based on pairing," in Proceedings of the 2000 Symposium on Cryptography and Information Security (SCIS 2000), Okinawa, Japan, January 2000, pp. 26–28.

[73] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing," J. Cryptology, 17(4), Sept. 2004, pp. 297–319.

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

[75] SHA-3 proposal BLAKE webpage. Retrieved [May 2015] from https://131002.net/blake.

[76] J. D. Bernstein, "The Salsa20 family of stream ciphers," Retrieved [May 2015] from cr.yp.to/papers.html#salsafamily.

[77] SERPENT webpage, "SERPENT A Candidate Block Cipher for the Advanced Encryption Standard," retrieved [May 2015] from www.cl.cam.ac.uk/~rja14/serpent.html.

[78] C. Petit, F. Standaert, O. Pereira, T. G. Malkin, and M. Yung, "A Block Cipher Based Pseudorandom Number Generator Secure Against Side-Channel Key Recovery," in Proceedings of the ACM Symposium on Information, Computer and Communications Security (ASIACCS '08), 2008, pp. 56–65.

[79] G. Anthes, "French team invents faster code-breaking algorithm," Communications of the ACM, vol. 57, no.1, January 2014, pp. 21-23.

[80] 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.

[81] The Legion of the Bouncy Castle webpage. Legion of the Bouncy Castle Java cryptography APIs. Retrieved [May 2015] from www.bouncycastle.org/java.html.

[82] SpongyCastle webpage, Spongy Castle: Repackage of Bouncy Castle for Android, Bouncy Castle Project (2012), Retrieved [May 2015] from rtyley.github.com/spongycastle/.

[83] V. Gough, "EncFS Encrypted Filesystem," stable release 1.7.4 (2010). Retrived [May 2015] from http://www.arg0.net/encfs.

[84] M. Riser, "Multiple Vulnerabilities in EncFS," 2010. Retrieve [May 2015] from: http://archives.neohapsis.com/archives/fulldisclosure/2010-08/0316.html.

[85] T. Hornby, "EncFS Security Audit," retrived [May 2015] from: https://defuse.ca/audits/encfs.htm.

[86] PhotoRec, Digital Picture and File Recovery. Retrived [May 2015] from: http://www.cgsecurity.org/wiki/PhotoRec.