Improving Android Database Security Using Elliptic Curve Integrated Encryption Scheme

Manendra Singh Rajpoot, Dr. Vivek Kapoor

Abstract— Have you ever give the impression of being at your Android applications and speculate if they are monitoring you as well? Whether it's a aggressive adware or even malware, it would be appealing to know if they are doing more than what they are thought to and if your personal information is uncovered.

In today's technical world the development is changing towards mobile now more than ever. The recent flow in popularity of smart handheld devices, including smart-phones and tablets, has given rise to new challenges in security of Personal Identifiable Information. In the course of smart phones, citizens make sensitive operations like mobile banking or online shopping: they also access their private documents and emails. Managers and executives make use of their corporate tablets for important projects and at meetings with customers. Moreover smart phones and tablets, due to their character, easily get stolen or lost. Therefore, it is very important that data and information stored in these devices are appropriately handled and not easily accessible to an eventually entrusted third party; at least those data that can be well thought-out as personal or private.

In this Report, I have offered a novel Database encryption method using elliptic curve cryptography to guard the detachable and constant storage on varied smart gadget devices running the Android platform. The planned encryption method leverages NIST certified cryptographic algorithms to encrypt the data-at-rest. Finally, we demonstrate that our approach is easy to install and configure across all Android platforms together with mobile phones, tablets, and small notebooks without any user perceivable delay for most of the standard Android applications.

I. INTRODUCTION

The Android platform implements security through a number of controls planned to care for the user. When an app is first installed, android checks the .apk file to make sure it has a valid digital signature to recognize the developer. After the .apk file is validated, Android checks the particular file created by the developer that specifies, between other items, what access an application wants to the system. A key component of the Android security model is that each application is assigned a unique Linux user and group ID which runs in its personal process and Dalvik VM.

These key mechanisms put into effect data security at the OS level as applications do not share memory, permissions, or disk storage. Also, most Android users have the choice to permit apps to be installed from nonmarket locations and to skip the digital signature check.

Problem :-

Mobile devices contain assets of personal and corporate data in a highly concentrated and transferable form. Criminals are generally practical and cyber criminals are no exception. If they can exploit one device that contains not only user names, passwords, and susceptible data about an individual, but also the same types of information about their employers, they will clearly target that chance. One of the most significant issues of modern computing systems is the prerequisite of sufficient security and privacy guarantees for the user data. However in the case of a mobile database application there are added security challenges due to the scattered nature of the application and the hardware constraints of mobile devices. Achieving a adequate level of security for such a platform is an important problem which has to be addressed.

Motivation :-

Bandwidth, battery life and memory are the key constraints one has to be in mind when deciding which type of Public Key Cryptosystem should be used in mobile environments. Wireless pagers, PDAs and mobile phones have constrained environments and are highly limited in processing power, memory and other resources. So an suitable public key system would be one that is more competent in terms of key sizes, computing costs as well as security.

Until now, no Public Key Cryptography system has a higher strength-per-bit than ECC. Small key sizes change into savings in processing power, bandwidth and memory. ECC has been the best selection in these cases.

In an situation where mobile data privacy is growing, this report will make it easier than ever for mobile developers to accurately secure their local application data, and in turn better shield the privacy of

their users. The data stored by Android apps confined by this type of encryption will be less exposed to access by malicious apps, protected in case of device loss or theft, and highly resistant to mobile data forensics tools that are ever more used to bunch copy a mobile device during routine traffic stops.

The intention of this project is to build up an competent and protected data storage system which is appropriate for android devices based on Password based Encryption using ECC protocols such that no third party would be able to view, modify or store any of the secret information, even if the attacker gets root access somehow. He/she shouldn't acquire hold on to the encryption key or data in plain text form.

Limited resources in the wireless devices put forward certain tradeoffs that need to be well thought-out for energy efficient secure communication systems. Usually, advanced security is achieved by using larger key sizes and stronger encryption algorithm. The higher security algorithm comes at the cost of increased computational time and energy utilization. However the battery power presented on the wireless devices is limited. Increasing the level of security would decrease the operation time of the device. The implications of providing security at different layers of the protocol would result in unusual delays.

Proposed solution :-

Elliptic Curve Cryptography (ECC) can be used as a tool for encrypting data, creating digital signatures or performing key exchanges. ECIES is the best known encryption idea in the scope of ECC, which is one of the most attractive recent cryptographic trends. Elliptic Curve Integrated Encryption Scheme is a amalgam encrytion scheme that works like stationary Diffie-Hellman followed by symmetric encryption. The scheme ECIES is poised of three algorithms: key generation, encryption and decryption.

ECIES provides some precious reward over other cryptosystems as RSA. RSA are well-known to be very costly with reference to calculation time and memory needs for larger key sizes. To resolve this problem, elliptic curve cryptosystems based can be used. An elliptic curve cryptosystem is an asymmetric cryptosystem based on the rigidity of the discrete logarithm problem in elliptic curve groups. With ECIES encryption, it is possible to encrypt data with a 160-bit key as safe as with RSA using a 1024-bit key. So ECIES encryption is the improved option compared to RSA when calculation time and available memory are limited, e.g. when using cryptosystems on smartcards.

Why password-based encryption is needed :-

There are different reasons why one would want to encrypt data in an Android application: to make certain that files exported to shared storage (SD card, etc.) are not easily reachable to other apps; to encrypt sensitive information (such as authentication information for third-party services) stored by the app or to make available some sort of a DRM-like scheme where content is only available to users who own the proper key to access it.

The Android SDK includes the Java Cryptography Extension (JCE) interfaces that grant easy access to common cryptographic operations, and all mainstream Android devices come with JCE providers that apply current symmetric encryption algorithms such as AES.

On the other hand, as in other systems, the harder part is not performing the actual cryptographic operations, but key management. If a key is stored all along with the encrypted data, or even as a file personal to the application, it is practically easy to extract it, mainly on a rooted device, and decrypt the data.

The same is correct for keys implanted in the application source code, even if they are to some extent obfuscated. There are in general two solutions to this problem: make use of a system service to protect the key, or don't store the key on the device at all, and have it entered each time access to protected data is required.

Android does offer a system key chain facility since version 4.0 (ICS), reachable via the KeyChain class. However, as discussed it can presently only be used to store RSA private keys and certificates. It is not common enough to allow secure storage of random user data, including symmetric encryption keys. That leaves us with the other alternative: do not store encryption keys on the device. However, symmetric encryption keys are long arbitrary strings of bits, and it cannot be anticipated that someone would actually memorize them, let alone enter them using an onscreen keyboard.

On the other hand, users are pretty known with passwords, and thus a way to generate strong cryptographic keys based on a humanly-manageable passwords is needed. There are standard and secure ways to do this, but let's first glance at some non-standard, and usually not secure, but quite common ways of producing a key from a password.

We will be using AES as the encryption algorithm for all examples, both for the reason that it is the current standard and is well thought-out highly secure, and because it is basically the only symmetric algorithm guaranteed to be available on all Android versions. Using symmetric encryption on Android is quite simple, but since a general purpose, system-level secure storage is not available, key management could be complex. One solution is not to store keys, but obtain them from user-entered passwords. Password strings cannot be used as symmetric encryption keys as is, so some

kind of key derivation is required. There are a few ways to derive keys, but most of them are not principally secure. To make sure encryption keys are both adequately random and hard to brute force, you should use standard PBE key derivation methods. Of those, the one at present regarded secure and available on Android is PBKDF2WithHmacSHA1. In short: when deriving a key from a password use PBKDF2WithHmacSHA1, a sufficiently long randomly generated salt and an iteration count appropriate for your app.

Key Derivation Solution :-

The suitable derivation of encryption keys is a critical part of any encryption system. Two well-known examples for Key Derivation functions are the PBKDF2 and SCRYPT functions.

Normally, KDFs are based on a common design. They take as input a random salt value, a passcode defined by the user, and/or a master key provided by a hardware component. While the salt input is requisite, dependence on the user input and on a hardware-protected key actually depends on the particular implementation. The selected input parameters for the used KDF directly manipulate the security of the derived keys and hence the overall security of the encryption system. Generally, the derived key is not directly used to encrypt the particular data, but only to protect the actual data encryption key. This two-stage approach is required to permit varying the passcode without the need to re-encrypt the protected data.

Challenges :-

1. Energy Consumption

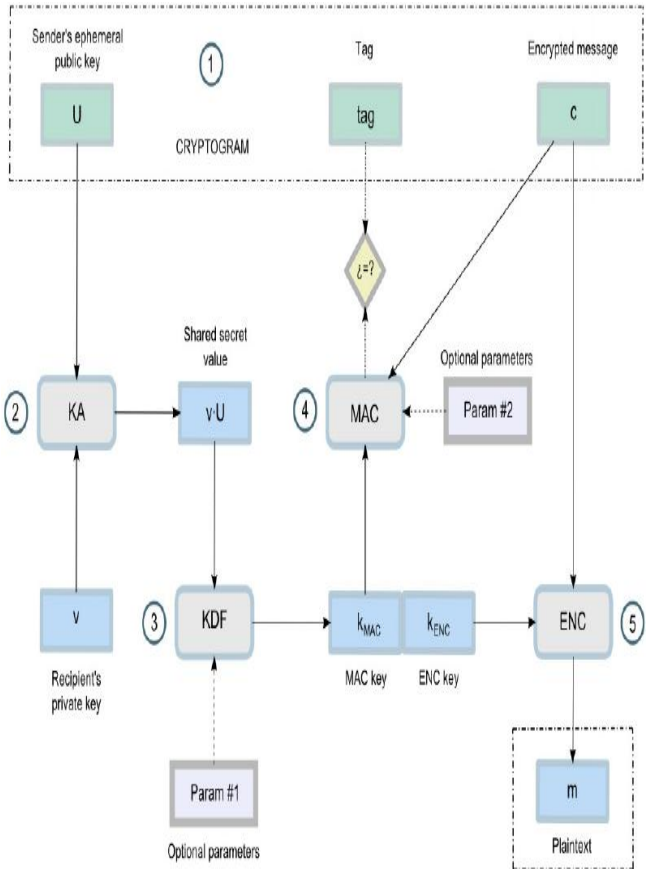
Resources in the wireless environment are restricted. The processor has limited capability and there is inadequate battery power available. The growing demand for services on wireless devices has hard-pressed technical research into finding ways to defeat these limitations.

Encryption, which is the backbone of security protocols, is computationally exhaustive and consumes energy and computational assets that are limited in wireless devices.

2. Bandwidth :-

When it comes to network bandwidth, the number one worry relates to the symmetric algorithm used for message encryption and Message Authentication Coding (MAC) for integrity scrutiny (this is not related to the choice of RSA versus ECC). Smaller embedded systems may start sessions more often, or the asymmetric authentication may be a larger fraction of the overall traffic and the size of the keys and signatures can make a variation. At the 128-bit security level, public keys and signatures are six-times bigger for RSA than for ECC. Private keys are 12-times bigger for RSA compared to ECC at the 128-bit security level. The key size usually has no impact on performance, but size matters when it comes to the cost of secure storage space of the keys.

ECIES encryption functional diagram



Result :-

Key Length	Time taken (in ms)
64	79ms
128	90ms
256	179ms
512	310ms
1024	563ms
2048	912ms
4096	1799ms

ALGORITHM:

Encryption:

STEP1: Produce public key and private key using ECIES.

STEP2: Encrypt data with public key and store in database.

STEP3: Create encryption key from application password using PBKDF.

STEP4: Encrypt private key by encryption key generated in step 3 and store it in database.

Decryption:

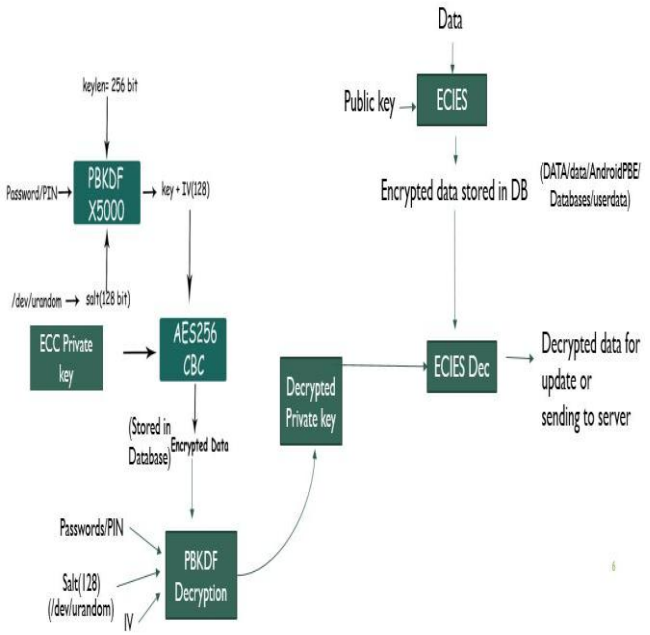
We need to decrypt the data only if user needs to update the data or uploads on server.

STEP5: Produce encryption key from application password using PBKDF. Same as step 3

STEP6: Regain encrypted private key from database and decrypt it using encryption key.

STEP7: Recover encrypted data from database and decrypt it using private key.

Architecture of proposed security model



Conclusion :-

Owing to the security issues, a good number new cryptographic protocols are moving away from RSA to elliptic curves. That change is happening even quicker in the embedded space where the ECC cost/performance benefits rapidly become important. Financial sector mobile apps are more sensitive than common purpose applications. Due to different platform availability in smart phones, these financial sector transactions and application usage requires high security and safeguard from any catastrophe and shield against threats. Therefore, in order to give security services to such applications requires a scalable, complete approach to mitigate consequences of threats in advance. We projected database encryption technology that provides analogous security level as full file system encryption supporting mobile phone. On the mobile phone with low performance, we preferred ECIES algorithm to reduce the computational complexity of public key algorithm. Mobile applications store their data typically in mobile phones internal memory which are always under danger. Salt is used to randomize the encrypted data so that attacker can't come across for any pattern. Also, our solution can easily be incorporated with any specific or general purpose mobile application and it can supply strong confidentiality, and integrity protection to application users.

ACKNOWLEDGMENT

References :-

1.BlueKrypt. "*Cryptographic Key Length Recommendation*," www.keylength.com, 2015.
2.National Security Agency. "Suite B Cryptography," 2014.
3.National Security Agency. "*The Case for Elliptic Curve Cryptography*," 2009
4. William Stallings, 'Cryptography and Network Security', Prentice Hall Publication, 1999
5.Bruce Schneier, Applied Cryptography, John Wiley & Sons Inc.
6.SEC2: Recommended Elliptic Curve Domain Parameters, Standards for Efficient Cryptography, Certicom Research, September 20, 2000
7.SSH Communications Security, 'Public Key Cryptosystems'
8.David Linden, Handbook of batteries, second edition. McGraw-Hill, Inc. 1994.
9.Eric Maiwald, Network Security: A Beginner's Guide, Osborne-McGraw Hill, 2001
10. Sushil Jajodia. Database security and privacy. ACM Comput. Surv., 28(1):129–131, 1996.
11.NIST SP 800-90, Recommendation for Random Number Generation Using Deterministic Random Bit Generators.
12.IETF RFC 2898 "PKCS #5: Password-based Cryptography Specification Version 2.0, September 2000.
13. FIPS 198-1, The Keyed-Hash Message Authentication Code (HMAC), July 2008
14.http://www.atmel.com/Images/Atmel-8951-CryptoAuth-RSA-ECC-Comparison-Embedded-Systems-WhitePaper.pdf
15.https://nelenkov.blogspot.ru/2012/04/using-password-based-encryption-on.html

Manendra Singh Rajpoot - Software engineer by profession, holds Bachelors' degree in C.S. SGSITS, pursuing Masters' degree I.T. with specialization in Information Security from IET DAVV Indore.

Dr. Vivek Kapoor – Senior faculty at Department of Information Technology, DAVV Indore. Holds Bachelors', Masters' degree and PhD. in Computer Science