Secure Android-based Mobile Banking Scheme

Hisham Sarhan
Al-Azhar University
Cairo, Egypt.

Ahmed A. Hafez
Military Technical College
Cairo, Egypt.

Ahmed Safwat
Al-Azhar University
Cairo, Egypt.

A.A. Hegazy
Arab Academy (AAST)
Cairo, Egypt

ABSTRACT
Smartphones and other mobile computing devices are being widely adopted globally [1].The increasing popularity of smart devices has led users to perform all their day to day activities using these devices [2]. Hence, M-banking has become more convenient, effective and reliable [3]. It is extremely necessary to provide the security services including; confidentiality, integrity, and authentication between the financial institutions' servers and the mobile device used by the customer, as their communications are through unsecured networks such as the Internet [4].Users' confidential information may be at risk due to fixed values-based security schemes, one level authentication, separate hard token-based authentication, hardware stealing, and Android-Based attacks. This paper specifies a comprehensive sought of how M-banking schemes can be assessed. Also it introduces a solution to mitigate most of these risks.

General Terms
Mobile Security, Digital Signature, Authentication, Android Operating System Security.

Keywords
Dynamic Initialization vector, Overlaid AES modes, Multi-Layer Authentication, Variable keys

1. INTRODUCTION
As smartphones started to replace the computer-based business applications and transactions, which marks the presence of confidential information, it becomes more vulnerable to cybercrimes. Mobile banking is the service that allows a mobile client to freely use his bank account for different services [5]. The main success factors of mobile banking are its convenience, ease of use, ubiquity and reliability. Our work, as described in this paper, enhancing the trials to keep private and sensitive information on modern Android devices, and communicate confidential information with a remote client in a security level compared to wired communication in spite of reported risks and threats to which the Android platform is exposed. The remainder of the paper is structured in the following way: Section 2 introduces background and related work, while Section 3 describes M-Banking security and capability considerations. Section 4 describes the solution we propose. And Section 5, finally, concludes the paper and indicates future work.

2. BACKGROUND AND RELATED WORK
The following subsections are brief background of technologies used in the proposal.

2.1 Wireless Public Key Infrastructure (W-PKI)
As the number of mobile users increases, the need of providing mobile phones with wireless internet services increases as well. The Security supporting wireless internet must be kept at same level as the wired internet security. But PKI (Public Key infrastructure) which is used for the security of E-Commerce in wired internet is not suitable for the mobile phone platform because of the fundamental limitations of performance such as limited memory size and limited computational capabilities. The security goals of PKI are authentication, Integrity, Confidentiality, and Non-Repudiation. A PKI binds public keys to entities, enables other entities to verify public key bindings, and provides the services of key management. The PKI relies on the following Components [6]

1- Certification Authority (CA)
2- Registration Authority (RA)
3- Certificate Distribution system or Repository
4- Certificate revocation list (CRL)
5- X.509 Public key Certificate.

2.2 Android Operating System
Mobile devices most commonly used and recent operating systems are Android, iOS, and Windows8. Through use of these operating systems, Android continuously and rapidly increases its popularity and market share. Based on the information provided by Google in March 2015, Google has shipped 1 billion Android-Based smart phones in 2014. In addition, the open-source nature of the Android platform, the ease of application development and submission process with the Play Store have made Android platform more attractive. However, the security risks and threats have increased and continue to increase more than for other mobile platforms as they are not open-source operating systems, such as Apple's iOS. Android is an open-source operating system that is built on a Linux kernel. It was developed under the leadership of the Open Handset Alliance (OHA) and Google. In this section, an Android OS overview is introduced.Fig.1 demonstrates the Android layered architecture which consists of 5 basic layers, and each layer has different program sets. Layers are Application, Application Framework, Library, Runtime, and Linux layers [3, 7]. The Android operating system has a basic security architecture that tries to secure user information and applications. The architecture provides a security model that adopts security in each layer and maintains flexibility in its design because of its open-source nature [2].

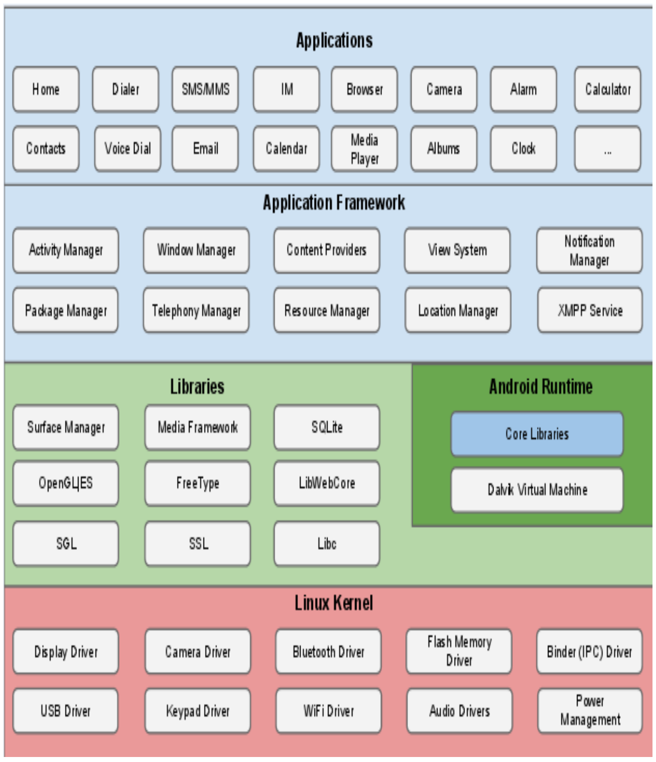

Fig 1: Android OS Basic Architecture

To achieve this goal, it supplies the following security features:

1- Powerful security mechanism on the Linux Kernel Level.
2- Sandboxing which means isolation of each application
3- Secure inter-process communication
4- Digital signature of applications
5- User approved and application specific permissions.
6- Approval for application stores

Android applications are generally coded in the Java programming language, and they run on DVM (Dalvik Virtual Machine). In addition, compilation from C/C++ language is available. Applications are installed from a single file with .apk extension. The basic structure of an Android application includes the following:

1- Android Manifest File
2- Activities
3- Services
4- Broadcast Receivers

As a default setting, applications have access to limited system resources. The permission mechanism handles this access management and checks whether these access processes are properly performed and do not behave maliciously. Restrictions are developed by using different techniques. In some cases, storage isolation is chosen for protection; in other cases, restrictions are performed based on the Permission List mechanism that secures sensitive APIs. A few of these protected APIs include Camera, Location (GPS), Bluetooth, Phone, Mic, SMS/MMS, and Network/Data (GSM/WCDMA/LTE and Wi-Fi) [8] [9]. Each API call in the Android operating system corresponds to permission in the manifest file (AndroidManifest.xml) that contains the list of permissions. When a user tries to install an Android application, the list of permissions that are mentioned in AndroidManifest.xml of the application is presented to the user. If user accepts this permission request, API calls become active. However, users can only allow or reject all of the permissions and do not have the power to select certain permissions. Allowing much unnecessary permission may cause security and privacy risks. Once the permissions are granted at the installation time, there is no way of changing these permissions. Furthermore, the permission model of Android OS does not support dynamic permission assignment. Establishing a secure communication link between Android-Based mobile phone and bank servers involving bank cloud faces some risks arising from attacking Android Operating System, Mobile phone platform or both. With regard to android operating system, Android's security mechanisms (both Android-specific and Linux-inherited) are insufficient and too coarse-grained to tackle this security issue. For example, an application granted Android application-level INTERNET permissions can listen on any port, create any type of socket, communicate with all protocols, and more. The file-permission mechanism protects files, but not from a root user [10]. Android platform application layer threats lie in the misuse of the permission model. Malwares are built on the poor technical background of most of android users, which makes no immunity against attacks. In [11] more than 1000 Android-Based applications were studied and authors tried to introduce the methodology by which developers use permissions in the Android OS. Based on study results, little number of permissions are used. They also found that the INTERNET permission was frequently used and suggested that a mechanism to control the usage of permissions need to be developed. According to their results, 60% of the applications only use the INTERNET permission. Some of recent related works that tried to mitigate the Android OS-related threats effects is well summarized in [2] which classified them as follows:

1- Operating System-Based solutions

In which the developers change the operating system architecture itself. In [12] researchers developed a system called (APEX) that allows user to choose some of or all permissions in the list. Researchers provided a user interface allowing the user to reject or allow any of the permissions listed on the interface during the application installation phase. This system achieves its goal by securing user from permission misus risks but it, extensively, requires a user with a good technical knowledge to decide which of those permissions are useful and which of them are harmful.

2- Permission-Based Solutions

These studies tried to provide statistical information by which user-developed applications can mitigate the harmful permission model usage in Android. In [13] Authors developed the VetDroid application that is able to analyze android applications, and give a report about permission model misuse. This doesn't propose a solution for preventing malware from permission misusing. While in [14] a new proposal has been developed. Authors developed two module-systems, first module is called Mr. Hide which can record interaction from user and then feed the second module, called Dr.Android, with decisions to rebuild the application considering first module output. For example, for INTERNET permission, it limits its use to limited URLs according to user feedback. However, the first module produces an extra 10-50% overhead on the system, and the consumed time to rebuild the application takes about one minute. The developed system causes overload as it uses mobile processing capacity while running other applications (in background) to rebuild the applications. Applications rebuild may cause in many cases slowing down application running time. In [2] authors recommended that, this application should have been tried on a wider range of android applications to show its reliability.

3- Source code-based solutions

These solutions consider the byte-coded files of applications. In [15] Authors proposed APPGUARD application that decodes the byte-code files and inspects policies, then, it regenerates the byte-code files of the applications to cope with the new policies. Rebuild causes in system overloading as the application files are rebuilt on the device itself. For instance, the time that is needed to rebuild the popular game of Angry Birds is 45 seconds, while Instagram requires 66 seconds and WhatsApp requires 57 seconds.

As for relation to Mobile phone platform, it provides more security, as it secures the storage from unauthorized use. In financial transactions, more than android application security services are required as it is not sufficient to rely on android OS security alone. The following researches contributed securing mobile platform using different strategies. In [16] Authors proposed A common application (for many banks connected to same Data base). User levels of security lies in TPA (Third Party Agent) which provides secret keys (passwords) of 6 digits to the user by sending it to the authorized email ,while Network level of security lies in using HTTPS which is used to connect the mobile data base to the server. However, Adding third Party is a cost inefficient and may cause security risks. Furthermore, TPA will send the secret password via email, which adds extra security risks. The Application doesn’t offer the user to change his/her password, this may cause problem as the user needs to receive an email on a periodic basis to renew the password. Sometimes, the password may not become secret as well. In [17] Sangram Ray and G.P.Biswas introduced the MHA (Mobile Home Agent) which performs all cryptographic operations in favor of mobile phone. MHA has the full responsibility on behalf of mobile user to store all the cryptographic information of mobile phone and performs all the operations required to get a certificate from CA. MHA generates an ECC-based public-private key pair for the mobile phone and sends it to RA as a certificate request message along with other relevant information to get the public key certificate of mobile phone from CA. After receiving the request, RA authenticates MHA and verifies the PoP function to be sure that the user possesses a private key corresponding to public key for which a certificate being requested. If the verification succeeds, RA sends the request to CA. CA checks whether the certificate request message is attested by RA , if so , CA creates the public key certificate, signs it , and publishes the certificate in its directory , and sends the URL to MHA , which in its turn , sends the URL to the Mobile user. However, adding a separate part “MHA“ may cause information leakage possibilities, adding a new part is more vulnerable to be stolen, lost or hacked, and even if the MHA is connected to the Mobile set , the mobile set itself may be stolen . Securing the mobile set with access control and encrypting the memory may reduce this risk. In case, the MHA is remotely connected, it will require additional authentication layer between mobile phone and MHA, which adds an additional throughput with additional time, and additional security risk as information leakage may be occurred. Furthermore, MHA is the main part of the transaction between mobile and the application server which may cause a bottleneck each time the mobile contacts the application server (or the bank). In [18] a one-time-password and a personal biometric have been combined with personal identification and password for verification while M-Banking. As the user presents the business transaction request, the server side will then generate an OTP and transfer to the default receiving equipment. If the input OTP (by the user) is correct, the user will be requested to capture fresh biometric data and upload it to the server side. Server side will compare and proceed if correct. Unfortunately, OTP transmission via internet makes it more vulnerable to risks as it may be intercepted or sent to a wrong recipient by the network operator by mistake. It may happen forcibly by DoS attacks that can congest all routes except the wrong maliciously intended route. And it may happen by mistake as well. Also, Biometric identification requires higher transmission rates, higher accuracy, as a small error may cause money being stolen and higher cost. It, also, requires very large storage devices as it will store images, videos, or audio files which have higher sizes than text files. In fact, same idea was discussed in [7] as the proposed solution depicts the use of biometric (sensor-based) identification to drive access control mechanism. In [19] authors introduce a design and implementation of a secure mobile wallet. Mobile Wallet is an application stored in mobile phones providing to subscribers the possibility to perform various mobile financial transactions. It stores the mobile wallet on the Java SIM Card (UICC).The wallet contains some security functions collected in “integrated security platform“ with Application-level PINs, Data Encryption ,FIPS 201 PIV authentication module and ,Asymmetric and symmetric cryptography. Two keys are generated, one is generated from the wallet called “Master Key “, another one is chosen by the user. The Master key is used for first key choice and reset. Master key is generated and transmitted by the Application issuer. The Encryption Key is generated by padding the 4 digits user key and hashing it, then combination with another padding, constant input, and constant Encryption key. However, this design uses Symmetric Techniques for signature, the symmetric algorithm has a fixed IV, and a full password text is used by the wallet to generate encryption keys which doesn’t meet key diffusion requirement. Finally, using RSA in mobile applications overloads the mobile set as it has a very long key 1024 bits to achieve a suitable level of security for M-Banking. In [20] authors deploy the PDE (plausible Deniable Encryption) in the Mobile communication after using in the desktop environment. It’s supposed that, the implementation of PDE is implicitly programmed into the android OS programming. Two types of PDE are exist, Stenographic PDE in which it encrypts files, but keeps them visible, and, Hidden Volume in which it encrypts files and hides them on the end of the storage. Less loss of data, less IO operations, faster, and higher security (not to reveal that, there are files the adversary cannot open it). Two modes of operations, Standard Mode in which it doesn’t reveal any secrets, user can login using the password (decoy password), and PDE Mode in which it applies the PDE to the entered data, user can login by true password. The entire disk is encrypted with a decoy key and formatted for regular use (the outer volume), then an additional file system is created at an offset within the disk and encrypted with a different key (true key); this is referred to as the hidden volume (i.e. the hidden volume use camouflages amongst the random data). When the user is coerced, he/she can provide the decoy key and deny the existence of hidden data. The Encryption data is XTS-AES. However, the design relies on the user to choose strong passwords to protect their encryption keys which makes the system more vulnerable to “offline dictionary attack “. Also, the salt is found in the Android encryption footer. The salt is used with PBKDF2. The salt cannot be stored in the hidden volumes as it is used to calculate the offset. Furthermore, after some statistical trials, it will be clear the regions with random data and ciphered (hidden) data, so the author solved it by using the encryption function (which depends on user-chosen password) to generate a random number to fill in the outer volume. The Encryption algorithm is AES with fixed IV, while it needs to have dynamic IVs to make it as random as possible.

3. M-BANKING SECURITY AND CAPABILITY CONSIDERATIONS

From section 2, and after induction, it is settled in mind that, M-Banking scheme should have : lower key sizes, lower processing and time requirements, higher data encryption level, higher data signature level, highly sophisticated authentication process, ease of use (i.e. it doesn't require any user's technical background), end-to-end security, need of communication through emails elimination, mitigation of the impact of viruses and malwares, withstand against attempted theft or loss, higher level of randomness and dynamicity, input-to-output diffusion, and withstand against most common attacks such as reply, dictionary, and search trial attacks.

4. PROBLEM DESCRIPTION AND PRPOSAL

Our proposal targets applications that are run on a smartphone device and communicate with remote service providers. One example is when a smartphone user needs to check his bank account or make some transactions remotely. Such applications exchange secret data that might be used for authenticating the smartphone user with the service provider in case of remote access or other business logic between them. Any remote information will be stored in the mobile device storage, and the application transfers the information during the transaction to the remote service provider. Keeping such data private is a security key purpose for both user and service provider. The proposed system consists of three parties, Android-Based Mobile device, Bank Server with local Database and its own cloud that can analyze SW applications and communicate with antimalware providers, and Bank provided SD Card. The following scenario summarizes the process:

User asks the bank to issue him a bank account. After completing the procedural requirements, user is asked for choosing some of Account privileges such as using M-Banking, or just internet banking. For M-Banking, bank is responsible for digital certificate issuance taking in consideration, at the first time, mobile IMEI (International Mobile Equipment Identification). Then, user will receive the SD Card by a secure delivery mean. The SD card will be loaded with all required policies, algorithms, initial keys including IV (AES Initialization vector) set, user chosen first level authentication key (password), some Bank digital certificate fields and finally, user's digital certificate all encrypted with AES-256-Dynamic IV. User is, also, required to make the SD card his default memory. On receiving the SD card from Bank side, user is required to login to Bank Mobile Website, within a limited period to download Bank mobile Application from the cloud. Fig.2 shows that, Cloud searches for the user's digital certificate on the mobile default SD card, if Bank cloud finds the certificate, it confirms the certificate validation from its database (whether it is on the correct IMEI or not, and it is still not revoked). If digital certificate and IMEI meet the stored data, then cloud will scan the mobile phone (this scan process will be taken place periodically to ensure non-existence of data leakage means, such as malwares). Cloud will rely on two approaches for the malware detection process, first is its own experience lists of malwares, and second is worldwide providers of anti-malware removal tools. If a harmful malware (high risk) is found, user is asked to remove it; otherwise cloud will not install the application. Actually, bank cloud application unit will categorize malwares into two categories

1- High risk Malware which has a direct limited less access permission to internet, SMS, location services, or any memory location
2- Low risk malware which doesn't have the above permission

Once malware clearance process is completed, bank cloud will allow mobile to download the mobile bank application which will carry out all bank-mobile interaction without extra technical requirements from the mobile user.

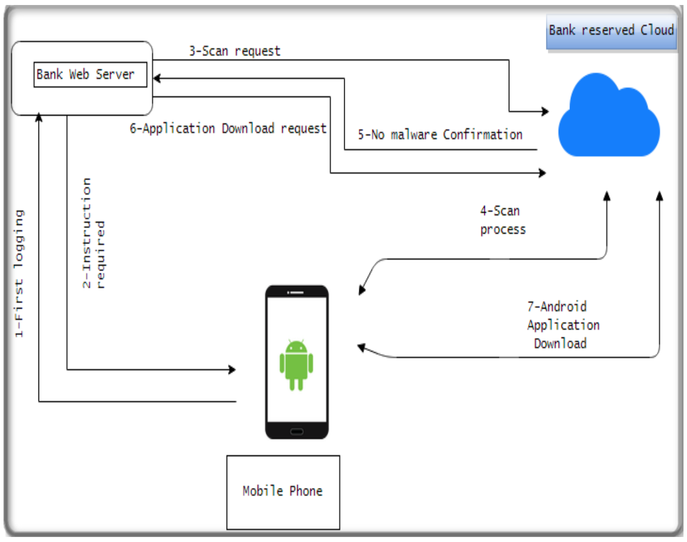

Fig.2 Android App. Installation process

Now mobile banking application is installed and user can login using his user name and first authentication level password that will allow him to, just, open the application without having any level of access authorization to the SD card or financial transaction. After that, as shown in fig. 3, user will choose a second level password for partially privileged use of the SD card within 5 minutes after first login to the application to secure the conjunction between first and second passwords. The second password will open only one user-chosen directory from which the third authentication key can be copied and then pasted to its filed in the application activity. Bank server will generate an OTP called OTP_third and put it in the user-chosen memory location on the bank provided SD, user accesses the location and copy the OTP_third to the application to fully open the SD card for the application.

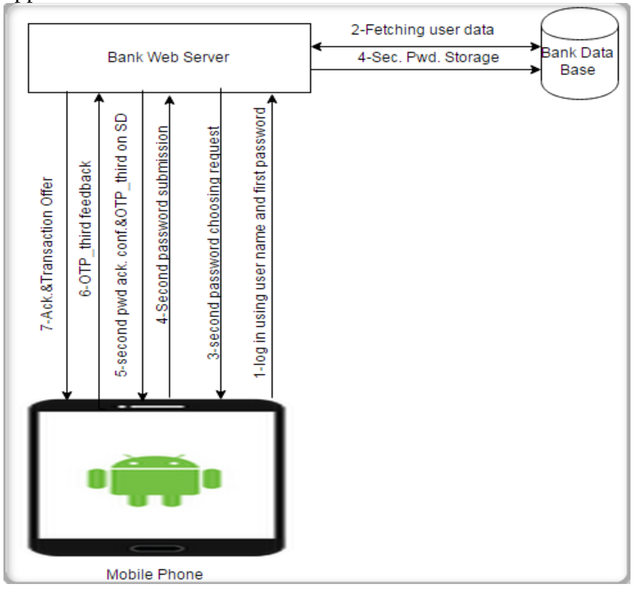
Bank Web Server
2-Fetching user data
4-Sec. Pwd. Storage
Bank Data Base
7-Ack.&Transaction Offer
6-OTP_third feedback
5-second pwd ack. conf.&OTP_third on SD
4-Second password submission
3-second password choosing request
1-log in using user name and first password
Mobile Phone

Fig.3First login to Bank Webserver

Up to now, Mobile phone user and Bank android application are well authenticated to each other. In case a transaction is being needed after user-application authentication process completion, the protocol diagram shown in Fig.4 is applied. In Fig.4, messages will be as following

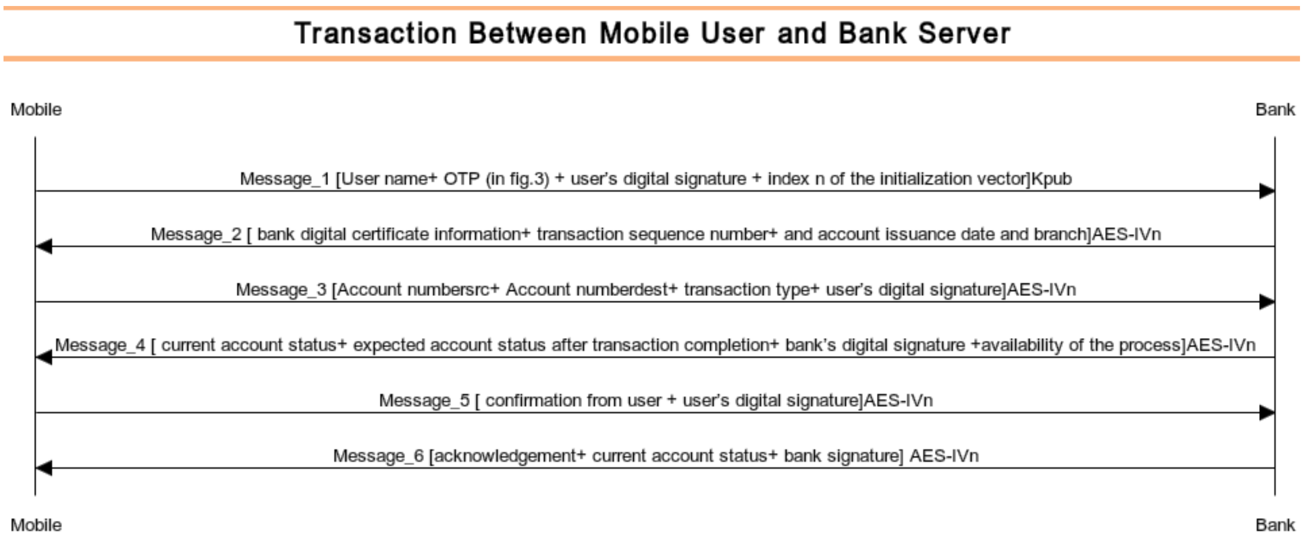
Transaction Between Mobile User and Bank Server
Mobile
Bank
Message_1 [User name+ OTP (in fig.3) + user's digital signature + index n of the initialization vector]Kpub
Message_2 [bank digital certificate information+ transaction sequence number+ and account issuance date and branch]AES-IVn
Message_3 [Account numbersrc+ Account numberdest+ transaction type+ user's digital signature]AES-IVn
Message_4 [current account status+ expected account status after transaction completion+ bank's digital signature +availability of the process]AES-IVn
Message_5 [confirmation from user + user's digital signature]AES-IVn
Message_6 [acknowledgement+ current account status+ bank signature] AES-IVn
Mobile
Bank

Fig.4 Proposed protocol Diagram

1- At bank, on reception of Message_1 [User name+ OTP (in fig.5) + user's digital signature + a randomly application chosen index n of the initialization vector all encrypted with bank's public key] It decrypts the message using its K_{pr}, then fetches the user information related to the given user name such as certificate and last transaction information, after that it calculates the user's expected OTP, and Message Digest from the signature by using his/her K_{pub}. Then bank asks for the user's certificate validation from CA through its cloud. This will be performed as the user certificate may be revoked during the period between application installation and first remote transaction due to security breakdown for example. Bank, then, compares the OTP and the digital signature to those sent from the user. If comparison succeeds, bank gets the user's IV_n from the IV set stored in the Bank's database and prepares Message_2 as shown in fig.4. Else, if the comparison fails, then un-encrypted rejection message will be sent.

2- At mobile phone, on reception of Message_2 [bank digital certificate information+ transaction sequence number+ and account issuance date and branch all encrypted using AES with IV_n where n is that one sent in message_1] application decrypts the whole Message by using the chosen IV_n. Then it checks the bank certificate information and account information with reference to those stored in the bank provided SD card. After that, it extracts the message digest by using the bank K_{pub}, compares all previous results, and then prepares Message_3 as shown in fig.4

3- At bank, on reception of Message_3 [Account $number_{src}$+ Account $number_{dest}$+ transaction type+ user's digital signature, all encrypted with AES using IV_n] bank decrypts the Message by using IV_n; Bank again confirms the user's signature. Then bank checks the transaction availability which includes (whether the required Service is in the bank portfolio and in the user basic list or not, Account has enough credit, and destination user account bank is in the network that the bank can transact with). If that above criteria is met, Bank prepares Message_4

4- At mobile , on reception of Message_4 [current account status+ expected account status after transaction completion+ bank's digital signature +availability of the process all encrypted using AES-IV_n] mobile confirms the bank again using the signature and by decrypting the message using the pre-agreed IV_n. Then mobile user gives the confirmation.

5- At bank , on reception of Message_5 [confirmation from user + user's digital signature all encrypted by AES-IV_n] bank completes the transaction, then prepares Message_6

6- At mobile, on reception of Message_6 [ack + current account status+ bank signature all encrypted with AES-IV_n] mobile confirms the bank, and then stores the transaction information in the SD card. The current log and last transaction information will be used in the third level authentication by a mean of combination with random numbers to ensure randomness and diffusion as shown in fig.5.

Algorithms and generators

1- Encryption algorithm is chosen to be AES-256 in two modes of operations, first is AES-256-CBC with dynamic IVs under the policy ,that will be discussed in the upcoming subsection, to be used to encrypt and decrypt messages between Bank server and mobile device, and ,second is AES-256-XTS encrypt the bank-provided SD-card. AES-XTS is designed for use in encrypting data stored on hard disks, and it works within the constraints imposed by disk hardware while keeping the security provided by the AES algorithm [21, 22].

2- Digital Signature algorithm is chosen to be ECDSA (Elliptic Curve Digital Signature Algorithm) as it emerges as an attractive public-key cryptosystem for mobile/wireless environments. ECC shows the same level of security with lower key sizes when it is compared to other algorithms such as RSA. It is worthy to note that a 160-bit ECC key has about the same level of security as a 1024-bit RSA key [17].

3- OTP generators is as shown in fig.5
OTP generator unit performs a random combination between the shown inputs and then hashing the output. The user chosen keys or passwords are not fully used to achieve diffusion as per fig.5 random digits selector randomly chooses four digits per time.

IV generation and policy

The IV set generation is performed at bank server, as the bank has higher storage capacity and processing capabilities than mobile equipment. The IV set is generated using a truly random generator to generate the first IV. Then other IVs are scrambled versions of the first IV. Each one IV set is valid for 3 years, with one different IV for each transaction as AES size is 256 and IV size is 256 as well, set will contain about 255 IVs assuming two transactions per week. While for a new certificate issuance or SD theft case a new IV set is delivered. After one IV set expiry, it will be reused considering the randomness basis.

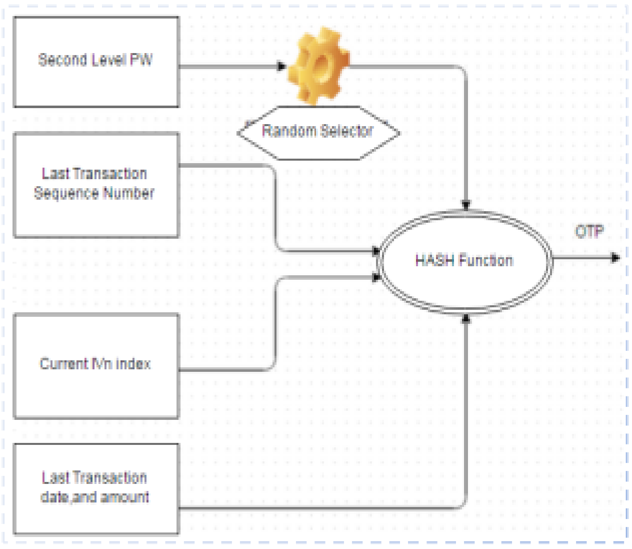
Second Level PW
Random Selector
Last Transaction Sequence Number
HASH Function
OTP
Current IVn index
Last Transaction date,and amount

Fig.5 OTP generator

5. CONCLUSION AND FUTURE WORK

Referring to section 3, the proposed solution theoretically covers most of those requirements and considerations. As it uses AES-256 with two modes of operation, AES-XTS for SD encryption, and AES-CBC with variable IV, it achieves a high level of data encryption. While ECDSA is used for Digital signature purposes which provide lower processing and key sizes. With respect to authentication, more than three levels of authentication are used as it asks for two user chosen passwords (one is known to bank representative and the other is anonymous) and bank generated OTP_third. Authentication phases ensure the mutual authentication between user and the APP, then between the APP and bank server. Bank cloud scans the mobile platform to ensure elimination and mitigation of malwares impact. Dynamicity of values, diffusion, and sequence numbers overcome most common attacks such as reply, dictionary, and praut force attacks. No keys will be sent via emails. At last, dynamicity of values, levels of authentication, bank authentication involvement, and encryption helps against theft or loss. The upcoming efforts will be concentrated on implementing the proposed solution and proving the system performance regarding time, security, and traffic.

6. ACKNOWLEDGEMENTS

Special thanks to Electrical Engineering department of Al-Azhar University for usual support during the research.

7. REFERENCES

[1] Adam Skillen and Mohammad Mannan “Mobiflage: Deniable Storage Encryption for Mobile Devices”, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, VOL. 11, NO. 3, MAY-JUNE 2014.

[2] Yuksel, Zaim, and, Aydin, “A Comprehensive Analysis of Android Security and Proposed Solutions” I.J. Computer Network and Information Security, 2014, 12, 9-20.

[3] Chang and Deng, “Secure OTP and Biometric Verification Scheme for Mobile Banking”, 2012 Third FTRA International Conference on Mobile, Ubiquitous, and Intelligent Computing.

[4] Majda, and Eihab, “Enhanced Model for PKI Certificate Validation in the Mobile Banking”, 2013, international conference on Computing, Electrical and Electronic engineering (ICCEEE).

[5] Narendiran, Rajendran and Albert, “PUBLIC KEY INFRASTRUCTURE FOR MOBILE BANKING SECURITY”.

[6] Cooper, Santesson, Farrell, Boeyen, Housley, Polk .2008. “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile” RFC 5280.

[7] Miriam, Ben-Av, and, Gerdov, “StoreDroid: Sensor-Based Data Protection Framework for Android”, Wireless Communications and Mobile Computing Conference (IWCMC), pages 511 – 517, Aug-2014.

[8] https://source.android.com/devices/tech/security/index.html.

[9] https://developer.android.com/reference/android/Manifest.

[10] https://developer.android.com/reference/android/Manifest. Permission.html.

[11] A Shabtai, Fledel, and Elovici.,” Securing android-powered mobile devices using selinux”, Ben-Gurion University. IEEE computer and reliability society, pages 36–44, May 2010.

[12] Mohammad Nauman, Sohail Khan, and Xinwen Zhang, “Apex: Extending android permission model and enforcement with user-defined runtime constraints”, In Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ASIACCS ‘10, pages 328–332, New York, NY, USA, 2010.

[13] Analysis for Vetting Undesirable” IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, VOL. 9, NO. 11, NOVEMBER 2014.

[14] Jinseong, Micinski, Jeffrey, Nikhilesh, Foster, Fogel, and Millstein, “Dr. Android and Mr. Hide: Fine-grained Permissions in Android Applications”, SPSM’12, October 19, 2012, Raleigh, North Carolina, USA.

[15] Backes, Gerling Hammer, and Styp-Rekowsky, ”AppGuard - Enforcing User Requirements on Android Apps A “Saarland University, Saarbrücken, Germany.

[16] Amol Bhatnagar, Shekhar Tanwar, and R.Manjula, “Secure Multiple Bank Transaction Log”, Inter. Journal of Research in Eng. And Technology IJRET, Apr-2014.

[17] Sangram Ray and G.P.Biswas, “Design of Mobile Public Key Infrastructure (M-PKI) using Elliptic Curve Cryptography”, Int. Journal on Crypt. And information security (IJCIS), Vol.3, No.1, March2013.

[18] Chang-Lung Tsai Chun-Jung Chen and Deng-Jie Zhuang, “ Secure OTP and Biometric Verification Scheme for Mobile Banking”, Third FTRA international conference on mobile, Ubiquitous and intelligent computing,2012.

[19] Hao Zhao and Sead Muftic, “Design and Implementation of a Mobile Transactions Client System : Secure UICC Mobile Wallet”, IJISR, International Journal for information security research , Volume 1 , issue 3 , Sep.2011.

[20] Adam Skillen and Mohamed Mannan, “Mobiflage: Deniable Storage Encryption for Mobile Devices”, IEEE, Transaction on dependable and secure computing, Vol11, No.3, May-June 2014.

[21] Morris Dworkin, “Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices“, NIST Special Publication 800-38E, January- 2010.

[22] I.G.Torrego 2009 Study of the IEEE Standard 1619.1: Authenticated Encryption with Length Expansion for Storage Devices. Master of Science in Communication Technology, Norwegian University of Science and Technology Department of Telematics.