Volume : 5 | Issue : 8 | Special Issue August-2016 • ISSN No 2277 - 8179

Secdata Collection: A Delay-Tolerant Security Data Collection App

Engineering

KEYWORDS: Index Terms- mHealth, salt, seed, Authentication

Jancy Manoharan C — PG Scholar, Computer Science & Engineering VedaVyasa Institute of Technology Karad, Malappuram

Ginnu George — Asst.Professor, Computer Science & Engineering VedaVyasa Institute of Technology Karad, Malappuram

ABSTRACT

Data collection is a very older concept which has marked a significant advancement in the history. Data collection is the process of gathering and measuring information on targeted variables in an established systematic fashion, which then enables one to answer relevant questions and evaluate outcomes. The data collection process is carried out for different purposes. The method of data collection also differs. The paper based data collection is an old method. Now days mobile electronic devices are used to collect data and that also give proper storage and data transfer to the remote database. When we are using this method of data collection to collect sensitive data for a large range of users, the major threat is that the lack of security. In this study it emphasis on health data which is consider as highly sensitive as it contain personal details. So here conduct a security ensuring mode to a data colleting app. Security is given to both stored and in-transit data. Here it also give an opportunity to checks whether agent is genuine or not. Altogether SecData collection is a data collecting app which can be use to collect sensitive data and it gives end to end data security. And the security framework is not include complicated encryption technique, but it include some special keys which are the main features of this framework.

I. INTRODUCTION

The new era of technology changes the world in different manner. Everything in day to day life is accomplished with the help of technology in easier way with high efficiency. mHealth is the combination of mobile computing technology with medical sensors and communication devices, creating solutions for improving health care. The mHealth concept is also related to that of electronic health processing, but while the latter is more focused on fixed computing facilities (e.g., desktop computers), the former aims to explore more intensively the advances in wireless communication, ubiquitous computing, and "wearable" device technologies .Several socioeconomic factors contribute to the interest in the mHealth trend. Examples include the broader availability of high-end mobile devices, the growth in coverage of mobile cellular networks, and the necessity of actively bringing adequate health care and support for people wherever they may be. This attention around the mHealth area is spread all around the globe, which has recently led the World Health. Organization (WHO) to develop surveys focused specifically on such solutions.

Applications surveyed include mobile telemedicine, decision support systems, solutions for raising treatment compliance and awareness, electronic patient records (EPR), and data collection systems for health surveys and surveillance, to cite a few. Among the conclusions drawn from such studies is the fact that, in the long run and after evaluation, mHealth solutions are expected to be integrated into and improve existing country-wide health strategies [3]. The use of mHealth solutions is a good way for an emerging country, to get improve health care by making utilities of technologies existing in the mobile market [4]. Indeed, specialized applications for health surveys and surveillance play a crucial role in such regions, providing a rich repository for decision making on the field of public health [3], [5]. Remote data collection of Primary Health Care indicators, such as family-related data, sanitary conditions, identification of common diseases in a given region, or tracking people with chronic conditions/diseases are the major category applications involved.

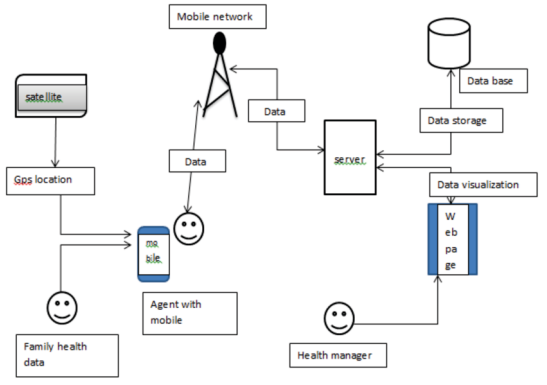


Fig1.remote data collection senario

A health team is responsible for data collection,they may include a medical persone(if needed) and a collection agent (who is responcible for a region) [6]. The collected data then pass to centralized server for futher use and discussion for accurate actions to be taken for better health care. Traditionally this was done with the healp of paper forms and this was really hard to work with since new updations made to the collected that is a task and handling a number of papers difficut. When using a mobile app for collecting data it will overcome this problems and sent to a database using the mobile network, increasing data reliability, and speed of decision making [7]. Despite its potential for effectively improving health and wellness, mHealth still faces many challenges for its widespread adoption [3]. One important concern refers to security: even though medical data are usually subject to a very strict legislation aiming to prevent unauthorized use or disclosure, many mHealth proposals do not employ robust security solutions to comply with such laws, hindering their ability to become real deployments [8], [9].

Aiming to tackle such issues, this paper presents SecData Collection, a security framework focused on highly sensitive data collection applications. The proposed framework relies on very lightweight mechanisms, securing the data exchanged with the server without requiring an extra security layer, transport layer security/secure sockets layer (TLS/SSL),except for the first time a user register in a new device. Aiming to analyze SecData Collection's performance, we also show how it can be integrated into a real data collection solution, Geohealth [6].

II. RELATED WORK

There are in the literature many frameworks for enabling generic data collection using mobile devices (for a survey, see [7]), and that can also be used by health applications. Despite their interest from an standardization point of view, the designs of such solutions usually provide only basic security measures, if any. For example, standards such as Open Data Kit [29] and openXdata [30] provide support to HTTPS and session authentication by means of username and password, while more advanced features such as secure storage and forward secrecy are not mentioned in their specifications. This reduced concern with security is somewhat understandable, since their design focus on helping the construction and management of data collection solutions rather than on protecting them. However, in scenarios that handle highly security-sensitive data such as medical information, an additional security layer becomes essential. Nonetheless, it is an unfortunate fact that strong security measures do not appear as one of the main concerns in many mHealth data collection solutions, such as GeoHealth [6], Epihandy [31], Borboleta [16], and Mobile Health Data Kit [7], to cite a few recent works. Actually, in our literature review of mobile data collection applications (both in the field of mHealth and more general scenarios), we were unable to find any solution displaying the flexible combination of forward secrecy levels provided by SecData Collection, or coping with all requirements discussed in Section II. Despite not being a majority, some interesting proposals for providing user/device authentication and data confidentiality in the context of mHeatlh applications do exist. Many of them focus on adding robust security mechanisms for EPR systems, considering scenarios in which data are exchanged inside a hospital or between health facilities. Examples include [32], which focus on allowing patients to asynchronously authorize a health professional to access their (encrypted) EPR data, and [33],which discusses how users could access their medical records from their homes using the GBA. There are also proposals for securely transferring medical information from/to the point of care [34], applying (offline) access control policies to the patients' data [35], and establishing authentication models suitable for health applications [36]. Even though such solutions share some features with the proposed SecData Collection framework, they usually focus on EPR security issues such as access control policies and securing in-transit data, not coping with many of the specific requirements of data collection systems (e.g., secure storage in scenarios with lack of connectivity).

Mechanisms for secure storage and end-to-end encryption appear in mHealth data collection solutions such as PopData [5]. However, one of the few thorough solutions in the literature that focus specifically on securing the whole mHealth data collection process is the protocol proposed in [15], which, together with the secure storage mechanism described in [37], forms an extension of the openXdata standard. The combined solution includes essential security features such as mutual authentication between user and server, encryption of stored data, and secure data delivery. Important requirements such as allowing devices to be shared and offline authentication of users are fully taken into account. Nevertheless, forward secrecy is not among the mechanisms provided, allowing attackers with physical access to the device to also access the stored data. Moreover, the protocol specification from [15] includes some apparently unnecessary operations. Namely, its registration process requires user and server to share not only a username and password, but also a secret key *Secret*. This secret key is not directly used for data encryption or mutual authentication, as could be expected, but rather employed for validating the server's public key upon the registration of the user in a new device. The public key is then used for encrypting a new symmetric key every time some data need to be downloaded from the server or uploaded to it. This profusion of keys not only makes the protocol more complex, but also defeats one of the main purposes of symmetric encryption: allowing two entities to communicate securely without the need of any preshared information. The net result of using public key encryption in this case is that it prevents attackers from tricking the device into Communicating with the wrong server, at the cost of a more computationally expensive protocol. SecData Collection not only adds forward secrecy to stored data, but also avoids the aforesaid issues altogether by almost removing the need for asymmetric cryptography, using shared key challenges for authentication whenever necessary. It also can leverage on mechanisms for establishing secure channels already employed in some applications, which could remove any need for especial-purpose public-key encryption protocols. Examples include the GBA [20], a very lightweight authentication mechanism for mobile networks, and HTTPS, which is more costly but remains widely supported and adopted by mHealthsolutions, including GeoHealth [6] and solutions based on the OpenRosa specification [38].

III. THE SECDATA COLLECTION FRAMEWORK

In this section, we describe the SecData Collection framework and its basic building blocks employed for fulfilling the requirements described in Section II.

A. User Registration

The registration process is the first process has to do for an agent to get authenticity to collect data. It is processes can only done in presents of internet connectivity. The agent registers with some valid information about him/her. Then the registration is valid only if the admin $\varepsilon S(A,B) = (\sum_{i=1}^{n}\sum_{j=1}^{n} A(i,j) \,\&\, \overline{B}(i,j)) \,/\, N$, (1) owing steps.

1) The mobile application generates a random *salt*, which is combined with the user-provided password by means of a key derivation function (KDF). The result is the master key *MK* = *KDF*(*salt* \\ *pwd*). The password itself can then be erased from the device's memory, since the registration process does not depend on its value from this point on. There is no restriction on the exact KDF function adopted for this process. A common approach is to employ the password-based key derivation function version 2 (PBKDF2) as defined in the PKCS #5 specifications [18], or more recent solutions such as Lyra [19]. The goal of such algorithms is to ensure higher security against brute-force attacks (also known as dictionary attacks)

2) The mobile application then establishes a secure connection With the server. Standard security solution such as TLS/SSL or the generic bootstrapping architecture (GBA) [20], [21] can be employed for this purpose. By using the protection of this secure tunnel, the mobile application sends the user identification *usr* together with the random *salt* generated in step 1.

3) The server computes the master key *MK* using the user's password and creates a random *seed* value. The value of *seed* plays an important role in providing forward secrecy to locally stored data, as further discussed in Section IIIC, but during the registration it is used simply for creating a challenge message.

4) The server makes a challenge to the mobile application: it encrypts the random *seed* using the previously computed master key *MK*, so that only a user who can compute the same *MK* is able to recover the correct *seed*.

5) The mobile application uses *MK* computed in step 1 for decrypting the value of *seed*. Then, it computes a shared key *K*0 as *K*0 = *H*[*seed*](*MK*) = *H*(*seed* \\ *MK*). As further discussed in Section IIIC, this key will be used for providing (strong) forward secrecy to data locally stored in the mobile device. At this point, however, it is used simply for computing the response to the server's challenge as *MACK*0 ('*user ok*' _ *usr* _ *salt* _ *seed*), which corresponds to the authentication tag of all previously exchanged variables together with a constant string.

6) The response is sent to the server.

7) The server computes the same *K*0 as the mobile application and verifies if the response provided by the latter is valid, which implies that the mobile device was able to recover the value of *seed* using the

Volume : 5 | Issue : 8 | Special Issue August-2016 • ISSN No 2277 - 8179

password input by the user. In case of success, the server stores *salt*, *seed*, and *MK*. These are associated with the corresponding user and identify him/her in the registered device, whose identification can also be stored for avoiding double registrations. The server also creates a positive assertion of the form $MACK0$ ('*serv ok*' _ *usr* _ *salt* _ *seed*).

8) The assertion generated in the previous step is sent to the mobile application.

9) The mobile application verifies the received assertion using *K*0 and the value of *seed* computed in step 5. If the verification is successful, this means that the user entered a legitimate password, whichwas not certain until this point. In this case, the application locally stores a) the value of the *salt* used during the protocol and b) an authentication token *Auth* = [*MACMK* ('*auth*')]*tsize* , which indicates that the user is registered in this mobile device and allows him/her to perform offline authentications afterward. Notice that the token is truncated to the system-defined parameter *tsize*, something that is further discussed in Section III-D.

Thereafter, the server can unregister any user simply by removing the entries for his/her username from its database.

10) The health admin will check whether agent requested is genuine or not, if so then make the registration conformed and allow them to collect data.

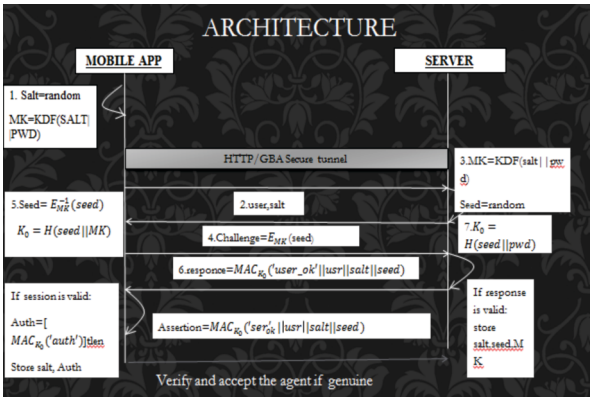


Fig 2.user registration

B. Offline User Authentication

Users can collect data by simply login to the app using the user id and password. After *usr* and *pwd* are provided by the user, the application employs the locally stored *salt* for computing the master key *MK* as described in Section IIIA, i.e., *MK* = *KDF* (*salt* \| \| *pwd*). This key is then used for generating a verification token *verif* = *MACMK* ('*auth*'), which is compared with the authentication token *Auth* locally stored. If [*verif*]*tsize* matches the value of *Auth*, the user is authenticated and can access the application. This process is illustrated in Fig.3Mobile app

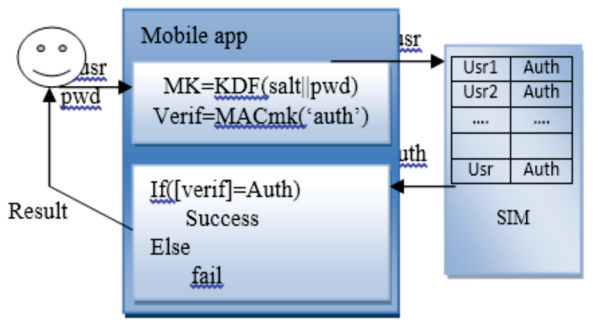


Fig .3 User authentication process(offline)

C. Secure Data Storage

After the authentication process is completed and the master key *MK* is computed from the user's password, SecData Collection generates keys that are used by an authenticated encryption algorithm *AE*() for protecting each form saved. The system can generate three types of keys, as described in the following. After such keys are generated, the master key *MK* is immediately wiped out of the device's memory.

1) No Forward Secrecy (Knofs): The *K*nofs key is computed directly from the master key *MK* as *K*nofs = *H* (0 \| \| *MK*) and is kept unchanged in the device's volatile memory until the application is closed. This key should be used for protecting forms that need to be easily recoverable during a session and that require no forward secrecy: its continuous availability in memory allows a user to promptly decrypt *K*nofs-protected forms without reentering his/her password, but, for this same reason, this key is revealed to attackers who are able to access the device's memory (e.g., by stealing the device while the application is still open).

2) Weak Forward Secrecy (Kwfs): The *K*wfs key provides weak forward secrecy, in the sense that an attacker who accesses the device's volatile memory, while the application still running is unable to decrypt any *K*wfs-protected form, but it does not protect data against attackers who discover the user's password. This key is computed at the start of the user's session as *K*wfs = *H*[*ses*](*MK*) = *H*(*ses* ,*MK*), where *MK* is the master .key and *ses* = 0is a session number renewed in every session (i.e., whenever a user reauthenticates im/herself).

3) Strong Forward Secrecy (Ksfs): The *K*sfs key provides strong forward secrecy, meaning that attackers are unable to decrypt any *K*sfs-protected form even if they discover the corresponding password or access the device's volatile memory while the application is still open. The reason is that this key is computed as *K*sfs = *K*0 = *H*(*seed* ,*MK*) right after the completion of the registration process and, analogous to *K*wfs, replaced by its hash value after being used to protect a form; in other words, the *is* th saved form is encrypted using *Kis* sfs = *His* (*seed* ,*MK*). Since the mobile device does not store the random *seed*, its value cannot be recovered using locally stored information, no matter if the password is known. This property also leads to the need of storing the next available *K*sfs in the device's nonvolatile memory after usage since, unlike *K*wfs , this key cannot be recomputed otherwise. In SecData Collection, after the *is* th form is encrypted, *Kis* +1 sfs is first encrypted with *K*nofs and then stored, which allows users to recover their next, still unused value of *K*sfs while preventing any other user sharing the device from doing the same. The improved security provided by *K*sfs is counterbalanced by the user's inability to modify the data after it is saved with this key, since the only entity that is able to decrypt and verify forms protected in this manner is the server itself. To do so, the server needs to be provided with 1) the value of the *salt* used for the corresponding user's registration, which identifies the value of *seed* and allows the computation of *K*0 ; and 2) the value of *is* corresponding to each form, which determines how many times *K*0 must be hashed for obtaining the correct *Kis*sfs .

It is worth noting that attackers who discover the user's password can violate the forward secrecy property of *K*sfs if they are able to recover the server's *challenge* (see step 4 of the registration protocol), whose decryption with the corresponding master key recovers the value of *seed*. Considering that this message is protected by a secure tunnel, however, doing so requires the attacker to break the tunnel's underlying protocol, which should be infeasible against technologies such as TLS, or compromise the server itself (who knows the value of *seed*), in which case the data would not be secure anyway. Nonetheless, if the attacker is somehow able to trick the user into creating a tunnel with him/her rather than with the legitimate server, a man-in-middle attack can be perpetrated and the security of the protocol is lost. Namely, the attacker can relay all messages between the user and the server, and then perform an offline dictionary attack on the keys established between them: the attacker can compute a

candidate *MK_* from *salt* and a guessed *pwd*, compute *seed_* and *K* 0 from *challenge* and *MK*, and then verify the guess by matching the user-provided *response* with *response_* = *MACK* 0 ('*user ok*' *_ usr _ salt _ seed_*). Avoiding such issue in the web may be a difficult challenge, since many users tend to ignore warnings about certificate errors when accessing websites [23], [24] and may end-up connecting to a fake server. Notwithstanding, such a threat should be more easily avoidable in the (presumably more controlled) scenario of data collection solutions, in which the application itself can utterly prevent users from making unsafe connections.

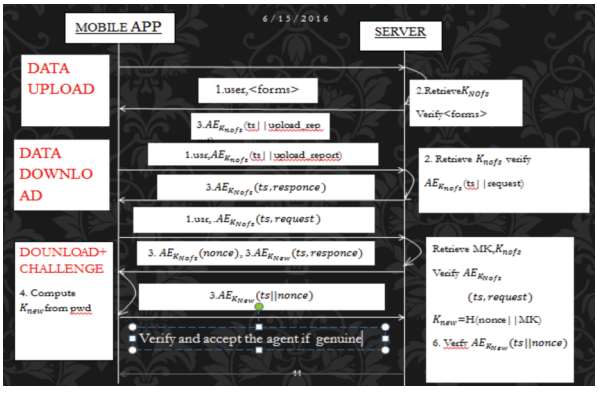


Fig4.data exchange with server

E. Data Exchange with Server Fig. 6 depicts the processes employed when the mobile device Sends / retrieves data to/from the server.

As discussed in Section IIIC, forms that are ready to be handed over to the server are authenticated when placed in the mobile device's local memory. Therefore, they can be delivered as soon as a communication channel is detected and without the user's intervention. The server will then be able to compute the required keys for decrypting and verifying the authenticity of the received data. This is done using information defined up on the user's registration (e.g., the master password) and/or appended to the forms (e.g., the value of is when Ksfs is employed). After a form is received and verified, the server must send a confirmation to the mobile device, which may then remove the corresponding data from its local memory. If this data delivery process is not performed inside a secure tunnel, the confirmation must be authenticated to prevent attackers from sending fake upload reports to the device; it also should contain a time stamp ts for identifying the request and avoiding replay attacks. As depicted in Fig. 6, in SecData Collection the upload report is protected using Knofs, allowing the device to transparently verify its authenticity. After a form is received and verified, the server must send a confirmation to the mobile device, which may then remove the corresponding data from its local memory. If this data delivery process is not performed inside a secure tunnel, the confirmation must be authenticated to prevent attackers from sending fake upload reports to the device; it also should contain a time stamp ts for identifying the request and avoiding replay attacks.

IV. ANALYSIS

SecData Collection was designed as a flexible solution for securing mHealth data collection systems, including lightweight security mechanisms that are useful in different application scenarios.

A. Tolerance to Delays and Lack of Connectivity

SecData Collection allows users to authenticate themselves in any device in which they have previously registered and then operate in a completely offline mode if required. Even though the registration itself requires connectivity, it must be performed only once per device, and it could be done before the users go to the field. After the data are collected, the forms are protected using one of the keys described in Section D, and can be delivered to the server as soon as a communication channel becomes available without need of a direct intervention from the user.

B. Protection Against Device Theft or Loss

SecData Collection supports data protection mechanisms with distinct security characteristics, allowing each application to adopt the required level of security against the capture of devices by attackers. The password itself is never left in memory, but used to derive different keys: Knofs , Kwfs , and Ksfs . Specifically, if only Ksfs is employed for protecting locally stored forms, attackers are unable to use them in offline dictionary attacks or to recover their contents after somehow discovering the password; Kwfs and Knofs are less secure in principle, but lead to better usability as they allow users to decrypt and modify stored forms if necessary. Finally, as discussed in Section E, SecData Collection includes mechanisms that limit the attackers' ability to retrieve information from the server even after stealing a device in which a legitimate user's session is still active.

C. Secure Data Exchange Between Mobile Device and Server

All data exchanged between server and mobile device are encrypted and authenticated, even in the absence of an underlying secure connection.

D. Lightweight and Low-Cost Solution

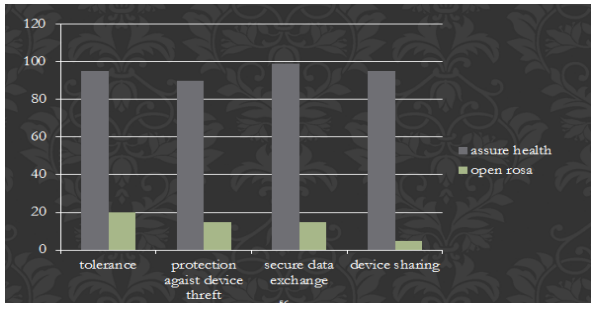
SecData Collection does not require any specific hardware and relies basically on lightweight cryptographic mechanisms, its (potentially) most expensive operation being the establishment of a secure tunnel during registration. Moreover, the protocols employed were designed to minimize the number of messages exchanged between server and mobile device: the most common operations (data upload/download) involve only one message from each side of the communication, while all other operations (registration and challenge issuing) involve at most two messages from each side.

E. Device Sharing

The proposed mechanisms allow legitimate users to register from any SecData Collection-enabled device while preventing users from accessing each other's data in shared devices.

F. Usability

SecData Collection allows users to access the system with a single credential. Moreover, it supports many configurable security usability tradeoffs. For example, the system can be configured to request the user's credential only once per session or, if desired, once again whenever a more sensitive operations is performed (e.g., accessing a locally stored form or downloading data from the server). Finally, data exchange with the server can be done without the user's direct intervention, allowing data to be quickly delivered whenever a communication channel is detected.



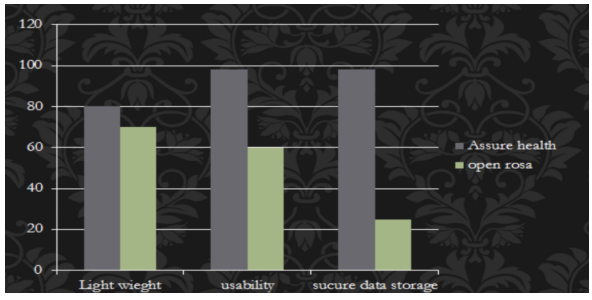


Fig 5,6 Evaluation with Open Rosa

V.IMPLEMENTATION

To assess the behavior of the SecData Collection framework in a real environment, we integrated the proposed mechanisms into the GeoHealth system [6], an Android-based application currently being used in the city of S˜ao Paulo as part of a governmental initiative for health data collection called "Family Health Program." This initiative involves teams of data collection agents responsible for assisting families in a well-defined geographical area, surveying several primary care conditions and promoting actions such as prevention, recovery, and rehabilitation. Partially filled forms are stored in the device's local memory so they can be filled later. After consolidation, the forms are put in a first-in-first-out queue and sent as soon as possible to the server. All collected data are geo referenced, providing health managers with a clear view of the population's conditions in the surveyed regions.

The original GeoHealth architecture uses password for protecting the access to the application. More precisely, before accessing the application, the user needs to send its password to the server to be validated and, in case of success, the password is stored in the mobile device's memory. HTTPS is used for securing all communications, including the password registration and data delivery. The data in this protected tunnel are not otherwise encrypted or authenticated. Even though this approach does not incur in any serious security issue for in-transit data, it leads to some undesirable overhead due to the repeated establishment of TLS/SSL sessions and it requires the password to remain in memory all the time. Moreover, no security mechanism is employed for protecting the information kept in the mobile device's memory while no communication channel is available. The SecData Collection-empowered GeoHealth system overcomes these issues in the following manner.

1) ***User Registration*****:** Even though the registration of a new user still employs HTTPS, the password is not sent in clear inside this tunnel but becomes part of the challenge response protocol described in Section IIIB. When compared to "plain" GeoHealth, this process adds some extra overhead before users can use a new device. Nevertheless, since this needs to be done only once and the whole as, process is very similar to the regular password registration, this burden is not significant in practice.

2) ***Secure Storage*****:** Partially filled forms are periodically saved by the system in an automatic manner. Hence, they are encrypted (but not authenticated) using *K*nofs, so they can be repeatedly accessed by the agents. Consolidated forms are not expected to be changed, since they are likely to be sent to the server automatically soon after being saved. Therefore, the system uses *K*sfs for encrypting and authenticating them. *K*wfs is not used in the system and, thus, it is not generated.

3) ***Data Exchange*****:** Data exchange with the server is performed without the prior establishment of an HTTPS channel, accelerating the delivery of consolidated forms. Downloading data from the server normally do not involve challenges issued by the server. The reason is that the policy adopted in GeoHealth when users request some data are already quite strict: the server has a list of families to be visited by each agent and usually prevents access to information not belonging to such families. Challenge issuing is, thus, limited to when an agent requests information about a number of families well above the average in the same day or in exceptional cases (e.g., unplanned emergency visits to families not assigned to the requesting agent). Namely, for the current average of six families visited per agent per day, a challenge would be issued when the agent requests information about the tenth family in less than 24 h.

A. Platform Characteristics

The platform used in the resulting integrated system is the Motorola Milestone 2, a reasonably high-end mobile device equipped with a 1-GHz processor, 8-GiB internal flash memory, 512 MiB of RAM, 3G connection, and a 5-MP camera. The implementation was done in Java using the Android Software Development Kit, which provides a set of application programming interface (API) libraries to build and test Android applications. The cryptographic algorithms employed were all taken from Spongy Castle (htt-p://r-tyley-.gith-ub.i-o/sp-ongyc-astle/), an Android repack of the Bouncy Ca-stle Java cryptography API (http://www.bouncycastle.org/java.html). Communications with the server are performed using a 3G connection with a nominal speed of 300 kbps.

B. Cryptographic Algorithms Adopted

The cryptographic algorithms used in the implementation are the following. We adopt PBKDF2 [18] as KDF for computing the master key from the password, using adequate parameters so that the total derivation time remains around 1 s. The underlying hash algorithm for PBKDF2 and other processes is SHA 256 [25]. Message authentication is performed using HMACSHA256 [26], while authenticated encryption is performed with EAX [27]. The underlying block cipher for all algorithms is the advanced encryption standard [28].

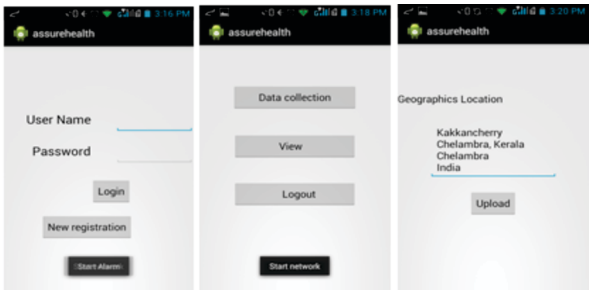


Fig.7 Screen shots

C. Pilot Application

Fig. 7 shows some screenshots of the client application's authentication process, which is the same for both registration (i.e., first-time usage) and any ensuing offline authentications. If this process is successful, the user does not need to re-enter his/her password until the application is closed, unless the server issues a challenge as previously discussed. After the data are collected, the corresponding forms are stored in the mobile phone's internal memory, in XML format. Partially filled forms are encrypted with *K*nofs . Forms that are completely filled and ready to be sent are authenticated and encrypted with the current value of *K*sfs and stored in a buffer.

Whenever the buffer is not empty, the application periodically searches for 3G connectivity until all forms are successfully delivered and erased from the buffer.

V. CONCLUSION

Security and reliability are most peremptory requirements for the success of systems that deal with highly sensitive data, such as medical information. Many existing mHealth applications are fail to protect both the locally stored and in-transit medical data gathered by them. To achieve this gap, this work proposes SecData Collection, a lightweight security framework designed specifically for this class of applications. SecData Collection ensures security for both stored and in-transit data. The system also persists for lack of connectivity, device sharing, and also it is transparent to users. The tools for checking connectivity and different services combined and used in the system. The experimental results when integrating SecData Collection into the Openrosa[6] show that it is possible to provide strong security for the data while introducing minimal overhead to the collection process. Finally, it is worth noting that even though SecData Collection was designed to prevent outsider rather than insider attacks (e.g., agents who simply copy old information into forms instead of effectively following their visitation schedule), since the proposed solution prevents outsiders from illegally accessing or tampering with the system's data, it gives managers the ability to identify and verify the registered agent and also misbehavior from agent occurs he can cancel agents registration accordingly.

REFERENCES

[1.] R. Istepanian, E. Jovanov, and Y. Zhang, "Guest editorial introduction to the special section on m-health: Beyond seamless mobility and global wireless health-care connectivity,"

[2.] S. Tachakra, X. Wang, R. Istepanian, and Y. Song, "Mobile e-health: The unwired evolution of telemedicine,"

[3.] WHO, "mHealth: New horizons for health through mobile technologies,"

[4.] L. Iwaya, M. Gomes, M. Simplicio, T. Carvalho, C. Dominicini, R. Sakuragui, M. Rebelo, M. Gutierrez, M. N¨aslund, and P. H"akansson, "Mobile health in emerging countries: A survey of research initiatives in Brazil,"

[5.] C. Hertzman, N. Meagher, and K. McGrail, "Privacy by design at population data BC,"

[6.] J. Sa, M. Rebelo, A. Brentani, S. Grisi, and M. Gutierrez, "GeoHealth: A georeferenced system for health data analysis in primary care,"

[7.] D. Shao, " A proposal of a mobile health data collection and reporting system for the developing world. "

[8.] A. Norris, R. Stockdale, and S. Sharma, "A strategic approach to mhealth,".

[9.] K. Patrick, W. Griswold, F. Raab, and S. Intille, "Health and the mobile phone,"

[10.] A. Sunyaev, J. M. Leimeister, and H. Krcmar, "Open security issues in german healthcare telematics,"

[11.] Infoway, "A 'Conceptual' Privacy Impact Assessment (PIA) on Canada's Electronic Health Record Solution (EHRS) Blueprint Version 2,"

[12.] Earth Institute. Barriers and Gaps Affecting mHealth in Low and Middle Income Countries: A Policy White Paper. Washington, D.C.: mHealth Alliance, 2010.

[13.] W. A. Kaplan, "Can the ubiquitous power of mobile phones be used to improve health outcomes in developing countries?"

[14.] J. G. Hodge, "Health information privacy and public health,"

[15.] F. Mancini, K. Mughal, S. Gejibo, and J. Klungsoyr, "Adding security to mobile data collection,"

[16.] R. Correia, F. Kon, and R. Kon, "Borboleta: A mobile telehealth system for primary homecare,"

[17.] J. Black, Authenticated Encryption.

[18.] B. Kaliski. (2000). PKCS#5: Password-Based Cryptography SpecificationVersion 2.0 [Online]. Available: http://www.ietf.org/rfc/rfc2898.txt

[19.] L. Almeida, E. Andrade, P. Barreto, and M. Simplicio, "Lyra: Passwordbased key derivation with tunable memory and processing costs,"

[20.] 3GPP, "TS 33.220 Generic Authentication Architecture (GAA), version 6.9.0,"

[21.] S. Holtmanns, V. Niemi, P. Ginzboorg, P. Laitinen, and N. Asokan, CellularAuthentication for Mobile and Internet Services.

[22.] D. Florencio and C. Herley, "A large scale study of web password habits,"

[23.] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor,"Crying wolf: An empirical study of SSL warning effectiveness,"

[24.] J. Engler, C. Karlof, E. Shi, and D. Song, "PAKE-based web authentication: The good, the bad and the hurdles,"

[25.] NIST. (2012, Mar.). Federal Information Processing Standard (FIPS 180- 4) Secure Hash Standard, Nat. Inst. Standards Technol., Gaithersburg, MD, USA [Online]: Available: csrc.nist.gov/publications/fips/fips1804/fips-180-4.pdf

[26.] Federal Information Processing Standard (FIPS PUB 198-1) The Keyed Hash Message Authentication Code

[27.] M. Bellare, P. Rogaway, and D. Wagner, "The EAX mode of operation: A two-pass authenticated-encryption scheme optimized for simplicity and efficiency,"

[28.] NIST, Federal Information Processing Standard (FIPS 197) Advanced Encryption Standard (AES), Nat. Inst. Standards Technol.

[29.] Y. Anokwa, C. Hartung, W. Brunette, G. Borriello, and A. Lerer. (2009, May), Open source data collection in the developing world. Comput. [Online]. 42(10), pp. 97–99. Available: see also: www.http://opendatakit.org/

[30.] OpenXdata, openXdata consortium (accessed on May 6, 2014).[Online]. Available: http://www.openxdata.org/

[31.] P. A. Bagyenda, D. Kayiwa, C. Tumwebaze, N. Frank, and M. Mark, "A mobile data collection tool Epihandy,"

[32.] T. Hupperich, H. L¨ohr, A.-R. Sadeghi, and M.Winandy, "Flexible patientcontrolled security for electronic health records,"

[33.] M. Shanmugam, S. Thiruvengadam, A. Khurat, and I.Maglogiannis, "Enabling secure mobile access for electronic health care applications,"

[34.] J. Mirkovic, H. Bryhni, and C. Ruland, "Secure solution for mobile access to patient's health care record,"

[35.] J. A. Akinyele, M. W. Pagano, M. D. Green, C. U. Lehmann, Z. N. Peterson, and A. D. Rubin, "Securing electronic medical records using attribute-based encryption onmobile devices,"

[36.] U. Sax, I. Kohane, and K. D. Mandl, "Wireless technology infrastructures for authentication of patients: PKI that rings," *J.*

[37.] S. H. Gejibo, F. Mancini, K. A. Mughal, R. A. B.Valvik, and J. I. Klungsyr, "Secure data storage for mobile data collection systems,"

[38.] OpenRosa, Openrosa consortium. [Online]. Available: ht-tp-://w-ww.d-im-agi.-com/collaborate/openrosa/