Adding Security to Mobile Data Collection

F. Mancini and K. A. Mughal
University of Bergen, Dept. of Informatics
Bergen, Norway
Emails: {federico.mancini,khalid.mughal}@ii.uib.no

S. H. Gejibo and J. Klungsøyr
Center for International Health
Bergen, Norway
Email: {samson.gejibo,jorn.klungsoyr}@cih.uib.no

***Abstract*— mHealth is having a profound and increasing impact on the delivery of medical and health services using mobile devices. However, many existing mHealth systems do not systematically address the security issues involved. As sensitive information is stored, exchanged and processed in these systems, issues like privacy, authentication, secure storage, accountability and permissions must be given top priority. In this paper we propose a protocol that provides end-to-end security, encrypted data storage and recovery mechanisms on mobile devices. We use openXdata as our reference mHealth system.**

I. Introduction

A number of systems already exist that allow data collection through mobile devices, especially in the health sector (an overview can be found at [10]). In order to establish a set of security requirements that should be fulfilled by such mHealth systems, we use openXdata [12] as reference application.

The openXdata system can be used to set up studies concerning virtually any type of data that can be collected through predefined forms (currently it is used especially in third world countries for collecting data in clinical trials, tracing the progress of large scale vaccinations or just monitoring student and teacher participation in schools). It consists of a server component that provides the tools to set up the studies, design the related forms, manage the users involved in the study and store and analyze the collected data. In the field, the data is collected by *collectors* which are entrusted with a mobile phone having the openXdata client installed on it. These collectors are given some area of responsibility and are in charge of downloading the forms for the current study, filling them with the required data, and uploading the data to the server.

It is easy to understand why effective security solutions are necessary when using such a system. The data that is collected can be extremely sensitive, and it should at all times be protected, both on the mobile devices and during its transmission to the server. At the same time, security should not compromise the usability of the application or the availability of the data that is collected. At the moment only rudimentary user authentication, but no real security, is in place in the openXdata mobile client. Even though HTTPS is supported, some project might not be able to use it because of practical constraints such as low budget, technical specifications, connectivity and usage of mobile devices, might require. This is why in this paper we propose a high level protocol that takes into account all these limitations, but still manages to provide basic security aspects like: authentication, secure data storage, confidentiality, data integrity, emergency data and password recovery and some degree of accountability.

The final goal is to develop a generic *Secure API* which can be easily integrated with existing mobile clients dealing with data collection. The idea is that developers can use the secure methods provided by the API to implement a secure layer on top of the existing application layer, both on the mobile and the server side. Furthermore, once the security layer has been implemented on the client, it should also be possible to externally configure the security level of the application to fit the particular needs of the specific projects deploying openXdata (or other similar systems). This extensibility would allow choosing which security provider to use, the strength of the cryptographic keys and which security aspects are required (only storage, both storage and communication, and so on).

The rest of the paper is organized as follows: first we explain in detail what security aspects we want to address and how the various limitations we mentioned influence our choices; we then describe the protocol in detail; and finally we discuss some related work and conclude with some topics for future work.

II. Working assumptions

A series of constraints need to be taken into consideration when designing a security solution for mHealth systems such as openXdata:

A. Low Budget Projects

Projects running on low budget could benefit from minimizing the cost of using certificate based security solutions. Such projects would out of necessity also deploy mobile phones with low-end specifications, that have low computational power and comparatively little memory. This needs to be taken into account in order to keep the computational burden of the cryptographic operations as low as possible.

B. Remote Working Locations

Many of the projects using openXdata are deployed in low-income countries like Uganda and Pakistan, where the infrastructure for mobile communication and Internet access may not yet be fully developed. Furthermore, much of the data collection might take place in remote locations or isolated villages, where the possibility of transmitting data through a mobile phone might be very intermittent and location specific.

This means that most of the data collection is done offline, and collectors must be able to authenticate themselves on the mobile phones without connecting to the server. Besides, even when some connectivity is available, the overhead due to the use of cryptography should be minimal in order to minimize both the cost of the data transfer and the possibility of transmission failure.

C. HTTPS Drawbacks

The HTTPS protocol is commonly used for secure client-server communication. However, the limitations described above excludes HTTPS.

HTTPS on mobile phones relies on a list of preinstalled root certificates from Certificate Authorities (CA). Such a list is not standardized across vendors or phone models and even the certificate format used may differ. A project might therefore require several CA issued certificates to guarantee compatibility, which translates into both a significant yearly cost and several distribution and management issues. Using free self-signed certificates is problematic as many phones do not accept them, and they do not guarantee the same security as those issued by a recognized CA.

Regardless of the costs and distribution problems, it is not given that HTTPS is the right solution for a system like openXdata. The HTTPS protocol is based on a long handshake to agree on client-server communication. This is unnecessary as it can be predefined and secure communication can be achieved with a lighter and faster protocol, as the one we propose in this paper.

D. Phone sharing

Another limitation is related to the fact that we cannot assume that each mobile phone is only used by a specific collector. One phone might be shared among collectors, each with their own account, and the same collector might be registered on more than one mobile phone. Phone sharing raises problems regarding privacy and access control, since most of the work is done offline and large amounts of sensitive data can be stored on a phone. Communication using SMS cannot be regarded as private with this constraint, making it difficult to communicate sensitive information to a collector. If email is not available either, it becomes problematic to issue, for example, new passwords.

E. Technical specifications of the mobile client

The current openXdata mobile client is developed using J2ME [13], in order to be compatible, in principle, with any phone that has J2ME support. This means that any security solution proposed must be compatible with any phone on which the openXdata client can run. Unfortunately not all J2ME specifications provide support for cryptographic APIs. There exists a package called SATSA-CRYPTO [14] which provides basic cryptographic primitives, but it is not supported by many phones yet and it provides only a limited type of cryptographic primitives. The most common alternative is to use Bouncy Castle [9], which is well-tested and constantly updated, supporting a wide range of cryptographic tools. However, the main drawback of Bouncy Castle is the large memory footprint of its libraries. Our aim is to provide a security solution that has an acceptable memory footprint and is compatible with the SATSA-CRYPTO package for compatibility with newer mobile phones.

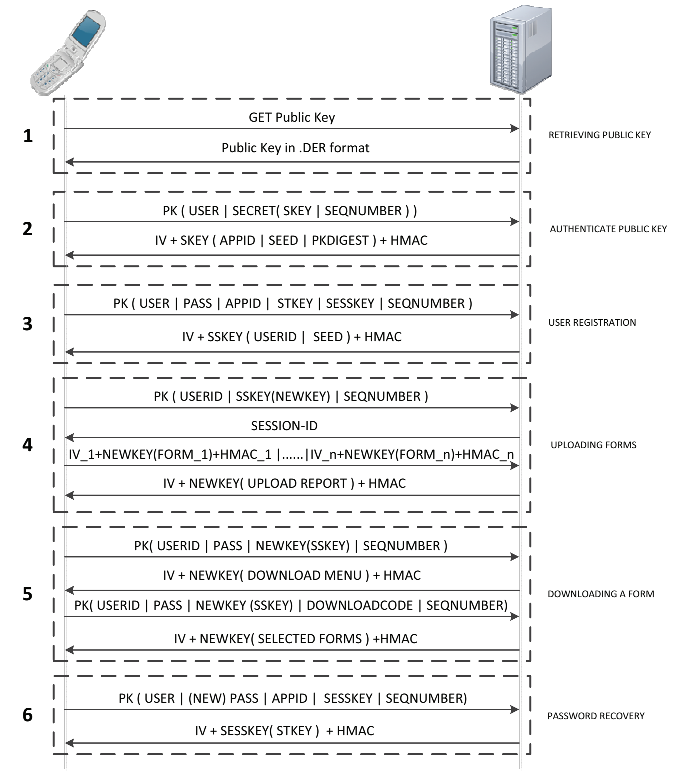
1
GET Public Key
Public Key in .DER format
RETRIEVING PUBLIC KEY
2
PK (USER | SECRET(SKEY | SEQNUMBER))
IV + SKEY (APPID | SEED | PKDIGEST) + HMAC
AUTHENTICATE PUBLIC KEY
3
PK (USER | PASS | APPID | STKEY | SESSKEY | SEQNUMBER)
IV + SSKEY (USERID | SEED) + HMAC
USER REGISTRATION
4
PK (USERID | SSKEY(NEWKEY) | SEQNUMBER)
SESSION-ID
IV_1+NEWKEY(FORM_1)+HMAC_1 |……|IV_n+NEWKEY(FORM_n)+HMAC_n
IV + NEWKEY(UPLOAD REPORT) + HMAC
UPLOADING FORMS
5
PK(USERID | PASS | NEWKEY(SSKEY) | SEQNUMBER)
IV + NEWKEY(DOWNLOAD MENU) + HMAC
PK(USERID | PASS | NEWKEY (SSKEY) | DOWNLOADCODE | SEQNUMBER)
IV + NEWKEY(SELECTED FORMS) +HMAC
DOWNLOADING A FORM
6
PK (USER | (NEW) PASS | APPID | SESSKEY | SEQNUMBER)
IV + SESSKEY(STKEY) + HMAC
PASSWORD RECOVERY

Fig. 1. Overview of the protocol. The notation "KEY(DATA)" is used to represent DATA encrypted with a KEY. In particular PK stands for public key and RSA encryption, while the others imply symmetric encryption.

A further challenge when using cryptography on a mobile phone is the generation of good random keys [3]. In fact, mobile phones do not have good sources of entropy, even though some studies are trying to overcome this problem [1].

Finally, one must also find the best possible compromise between key sizes and available computational power in order to create a secure, but usable application. In this paper we propose a solution that strikes a balance with regard to these issues.

III. The Protocol

Conceptually we can think of the protocol as being divided into three main parts: Authentication and Registration (Step 1, 2 and 3 in Figure 1), Secure Storage, Login and Password Recovery (the first two are described together in the registration step, and the third is shown in Step 6 in Figure 1) and Data Transfer (Step 4 and 5 in Figure 1).

The first part can be thought of as the initialization of the application on a mobile phone before it can actually be used to (securely) collect data. The second part deals with the security while working offline, hence how to authenticate a user locally on the mobile phone, and how to keep the stored data secured. The last part deals with the data exchange between mobile and server.

The general idea behind the protocol is that, even though HTTPS might not be a viable option, we still want to use public key cryptography for critical information exchange (i.e. user credentials and symmetric keys), and symmetric cryptography, which is more efficient, for the protection of the data both on the mobile phone and during upload/download. Also, we would like to optimize efficiency of the cryptographic operations on the device, and minimize the communication overhead. For this reason, the protocol is stateless, i.e., all transactions consist of a single request and a single response. Any other solution would require the use of session-ids, which would have to be exchanged in clear since, unlike SSL where the whole communication channel is encrypted, we secure only the application data. We make an exception for the upload of the collected data as we explain later.

Not using session-ids means also that server never gets a confirmation to its responses. One of the challenges is therefore to design a safe and efficient way to re-send a request to the server if, for some reason, the response was lost or corrupted. To avoid replay attacks in this context, we always add a sequential number to each request, making it unique and not reusable. The other solution for this problem, using time stamps, would require keeping the client and the server synchronized, which is not feasible in our scenario.

For symmetric cryptography we propose to use the AES algorithm in CBC mode, with keys of 128 or 256 bits and random unique IVs (Initialization Vectors) for each new encryption. In addition an HMAC (Hash-based Message Authentication Code) should be added to guarantee the data integrity and some degree of accountability.

We assume that the application has been configured and installed properly on the mobile device. The individual steps of the protocol shown in Figure 1 are discussed below:

1) **Retrieving server public key:** The first time the application is opened, the server URL is entered to request and retrieve the public key.
2) **Authenticating the public key:** The public key needs to be authenticated, since we cannot rely on certificates. This is done by challenging the server to decrypt a secret key. To be able to perform the decryption, the server must use a shared secret that was distributed in advance to the users together with their user name and password. If the server can decrypt the secret key, it will be able to encrypt its response consisting of a unique application id to identify the specific device running the application in future requests, and a seed to improve the quality of the keys generated by the random generator on the mobile phone. The public key digest is used as a proof that the server sending the response is the same one that initially supplied the public key.
3) **Registering a user:** After server authentication, a user must create an account on the mobile phone. Apart from sending the user name and password in the request, a storage key is also sent that will be used to encrypt the data on the phone, and a session key that will be used to encrypt the traffic. If the credentials are correct, the server stores the keys and returns a unique user id to identify the user on the specific application instance.
All cryptographic information pertaining to the user after registration is stored in a user specific Record Store, to which only the Secure API has access. The data in the forms, instead, is encrypted by the Secure API, but managed and stored by the client application.
After registration, the user is asked to choose a password to logon to the mobile phone (thus creating a mobile password). This password is used to generate a key through a Password Based Encryption (PBE) scheme [8], which is in turn used to encrypt the storage key of the user on the phone. The storage key itself is used to encrypt all other data about the user. This increases the independence between security and application layer, and allows multiple user to be registered on the same device, without compromising accessibility or privacy.
The login procedure is based on the successful decryption of the storage key. A key is generated from the password using the PBE scheme and the salt created at the registration step, and used to decrypt the storage key. Since the storage key is encrypted together with its digest, it is possible to check whether the decryption was successful, and therefore authenticate the user. Using two passwords reduces the risk that an attacker can gain access to the data on the server if the device and/or the mobile password are stolen. In fact, the password to access the server is the same as the one used for registration and is not stored in any form on the mobile. However, completely protecting against a brute force attack on the data stored on the phone is practically impossible (see also [4]).
4) **Uploading data:** In order to optimize the upload procedure, the forms that have been completed are encrypted directly with the session key, rather than the storage key, before being stored. To initiate upload, the client asks the Secure API to open a connection to the server by authenticating the user. The Secure API receives a session id from the server, and opens a new connection with that session id. This connection is returned to the client application, that simply retrieves the already encrypted forms from the Data Store and sends them to the server. Although this approach might expose the connection to session hijacking attacks, we rely on the fact that data is encrypted and that only upload is possible in such sessions, limiting any potential damage if an attacker gains access to the session key. Risk is further minimized by setting the duration of the session appropriately.
5) **Downloading data:** A flawed download procedure can compromise the server, where as a flawed upload procedure can only compromise the data on a single phone. To avoid compromising the server, the download procedure

does not rely on session ids. Instead, the encrypted credentials of the user are added to each request to the server.

6) **Recovering an account:** The user password (the one used in the registration step) can be used to reset the mobile password. If the mobile password is lost, the user can still be authenticated on the server and can retrieve the storage key that was stored there at the registration step. The storage key can then be encrypted with a new password on the mobile phone, and replace the one encrypted with the forgotten password. This way all the data on the phone is accessible again. If the server password is lost, than a new password must be issued at the discretion of the server/administrator. Thus issuing a new user password does not affect the use of any of the mobile phones where the user is registered. Only the server needs to know about the change, and the mobile device can be normally accessed and used offline.

IV. Related Work

Currently most mHealth systems for data collection do not systematically address security issues and even SMS is widely used. There are mHealth systems that use HTTPS for the communication between mobile devices and server e.g. EmitMobile by Cell-Life (based on openXdata) [2] and EpiSurveyor [5] by DataDyne (based on JavaROSA mobile [11]). These represent commercial solutions, which use a centralized proprietary server, but there is still no encryption of data stored on mobile device. The OpenRosa consortium [10] is working to define general standards for mobile data collection practices and a particular reference J2ME implementation [11], including security. However, at the present time, the only adopted standard proposed as alternative to HTTPS is the Basic and Digest Access Authentication defined in [6].

A protocol similar to the one we propose is presented in [7]. In this paper, the authors also consider the problems of authentication, confidentiality and secure storage, but starting from different working assumptions. In their scenario, the application would be customized and packaged for each user, so that all the necessary shared secrets and keys would be already bundled with the JAR file of the application. The application would also be installed directly on the user phone by an administrator. This eliminates most of the problems we face. In addition only symmetric encryption is used. Finally, the secure storage problem is reduced to simply encrypting the user keys that came with the application, and no real password based encryption is used.

V. Conclusions

The protocol described in this paper addresses security concerns regarding low-end mobile devices used for collecting sensitive and personal data. However, we now need to provide a proof of concept for our protocol. The implementation of a prototype is still ongoing and no extensive tests have been performed. There are a number of issues that will have to be dealt with and clarified once the prototype is complete.

First of all, a more systematic security analysis of the protocol must be performed, but this is difficult to do before an actual implementation is in place and all the protocol steps have been tested. In fact, some of the cryptographic operations might prove to be too computation intensive for some lower-end phone (for example, generation of an encryption key from the user password), and some adjustments might be required (for example, adjusting the number of iterations for key generation).

Finally, the protocol should be deployed and tested in real projects for further improvement. Extensive performance tests should be run to analyze the impact of cryptography on the device performance and on the communication channel. We must also test the performance and overhead of each step of the protocol in different scenarios and on different devices. This will allow us to optimize the implementation and to minimize any adverse effects by the security layer on the application.

Once a reference implementation of the protocol is completed, an API should be designed to allow smooth integration with other existing data-collecting mobile clients, starting with the openXdata mobile client. The challenge will be to take into account possible configurations of the security concerns, both from the developer point of view and the end user.

References

[1] J. Bouda, J. Krhovjak, V. Matyas, and P. Svenda, "Towards true random number generation in mobile environments," in *Identity and Privacy in the Internet Age*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2009, vol. 5838, pp. 179–189.

[2] Cell-Life, "Emit," Accessed Mars 2011. [Online]. Available: http://www.emitmobile.co.za/

[3] S. Crocker and J. Schiller, "RFC 4086 - randomness requirements for security," 2005, Accessed Mars 2011. [Online]. Available: http://www.ietf.org/rfc/rfc4086.txt

[4] T. Egeberg, "Storage of sensitive data in a Java enabled cell phone," Master's thesis, Hgskolen i Gjøvik, 2006.

[5] Episurveyor, Accessed Mars 2011. [Online]. Available: http://www.episurveyor.org/

[6] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart, "RFC 2617 - HTTP authentication: Basic and digest access authentication," 1999, Accessed Mars 2011. [Online]. Available: http://www.ietf.org/rfc/rfc2617.txt

[7] W. Itani and A. Kayssi, "J2ME application-layer end-to-end security for m-commerce," *Journal of Network and Computer Applications*, vol. 27, no. 1, pp. 13–32, January 2004.

[8] B. Kaliski, "RFC 2898 - PKCS #5: Password-based cryptography specification," 2000, Accessed Mars 2011. [Online]. Available: http://www.ietf.org/rfc/rfc2898.txt

[9] T. Legion Of the Bouncy Castle, Accessed Mars 2011. [Online]. Available: http://www.bouncycastle.org/

[10] Open Rosa, Accessed Mars 2011. [Online]. Available: http://www.openrosa.org

[11] ——, "Javarosa," Accessed Mars 2011. [Online]. Available: http://www.javarosa.org

[12] OpenXData, Accessed Mars 2011. [Online]. Available: http://www.openxdata.org

[13] Oracle, "Java ME reference," Accessed Mars 2011. [Online]. Available: http://www.oracle.com/technetwork/java/javame/documentation/index.html

[14] ——, "Security and trust services API for J2ME (SATSA)," 2006, Accessed Mars 2011. [Online]. Available: http://java.sun.com/products/satsa/