Securing the Access to Electronic Health Records on Mobile Phones*

Alexandra Dmitrienko[1], Zecir Hadzic[1], Hans Löhr[1], Ahmad-Reza Sadeghi[2], and Marcel Winandy[1]

[1]Horst Görtz Institute for IT-Security (HGI), Ruhr-University Bochum, Germany
{alexandra.dmitrienko,zecir.hadzic,hans.loehr,marcel.winandy}@trust.rub.de

[2]Center for Advanced Security Research Darmstadt (CASED),
Technische Universität Darmstadt, Germany
ahmad.sadeghi@trust.cased.de

Abstract Mobile phones are increasingly used in the e-health domain. In this context, enabling secure access to health records from mobile devices is of particular importance because of the high security and privacy requirements for sensitive medical data. Standard operating systems and software, as they are deployed on current smartphones, cannot protect sensitive data appropriately, even though modern mobile hardware platforms often provide dedicated security features. Current mobile phones are prone to attacks by malicious software, which might gain unauthorized access to sensitive medical data.
In this paper, we present a security architecture for the protection of electronic health records and authentication credentials that are used to access e-health services. Our architecture is derived from a generic solution and tailored specifically to current mobile platforms with hardware security extensions. Authentication data are protected by a trusted wallet (TruWallet), which leverages trusted hardware features of the phone and isolated application environments provided by a secure operating system. A separate application environment is used to provide runtime protection of medical data. Furthermore, we present a prototype implementation of TruWallet on the Nokia N900 mobile phone. In contrast to commodity systems, our architecture enables healthcare professionals to securely access medical data on their mobile devices without the risk of disclosing sensitive information.

1 Introduction

The usage of mobile phones as multi-purpose assistant device in healthcare has been proposed in several application scenarios. Its usefulness is derived from its mobility and flexibility, i.e., today's smartphones offer appropriate computing and storage capacity allowing the realization of various applications that can be used basically from everywhere. For instance, healthcare professionals can use a

* An earlier version of this paper has been published in [11].

mobile phone to download and share electronic health records of their patients [9]. In other scenarios, patients use their mobile phones to provide personal health data, e.g., taken from additional bio-sensors, to a medical information and diagnosis system [17].

While smartphones are very flexible and cost-efficient computing devices, they generally do not offer sufficient security mechanisms to protect the data they operate on. This is mainly due to the architectural shortcomings of their operating systems, which are derived from the same (security) architecture as desktop operating systems. Typical examples are Google Android [6], Apple iOS [7], Symbian [35], and Windows Mobile [28]. Although, some of them provide more sophisticated security mechanisms than their desktop counterparts, e.g., application-oriented access control in Android [16], they still suffer from fundamental security problems due to their large code base and complexity, lacking of strong isolation of applications (*secure execution*) and insufficient protection of stored data (*secure storage*). Recent attacks on smartphones demonstrate their vulnerability [1,19,39]. But the secure operation of a mobile phone is an important aspect when a user is working with security and privacy-sensitive data such as personal health records on the device.

Especially in healthcare telematics infrastructures, the end-user systems of health professionals have been identified as an insecure and less specified component [34]. Malware on the user's computing platform could steal passwords that are used to access healthcare information systems, manipulate data such as medical prescriptions, or eavesdrop on and copy private data such as personal health records. While the connection of stationary desktop systems to the healthcare telematics may be protected by additional secure hardware network components like, e.g., special firewalls and gateway routers, the situation gets worse when mobile phones are used. Due to their mobility and changing connectivity (wireless LAN or GSM network), mobile phones may usually only use Virtual Private Network (VPN) technology to secure the connection. But the necessary credentials, like user passwords and VPN keys, are not sufficiently protected against malware on the device, and, hence, could be accessed by unauthorized parties.

However, modern smartphone hardware offers advanced security functionality, which are embedded in their processors, but generally not used by the mainstream mobile operating systems. For instance, ARM TrustZone [37] and Texas Instruments M-Shield [8] offer secure boot[1] functionality, secure storage and secure execution environments for security-critical functions, which are isolated based on hardware mechanism from other processes running on the phone.

Contribution In this paper, we propose a security architecture for accessing e-health services on mobile phones. We present the combination of efficient solutions that current technology can offer on mobile phones for the secure handling of accessing and processing of security-sensitive data such as electronic health records. In particular, we propose (i) a security framework to create a secure run-

[1] Secure boot means that a system terminates the boot process in case the integrity check of a component to be loaded fails.

time environment for medical applications, and (ii) specific tools that protect the authentication of users and their mobile devices to e-health servers.

In our security framework, we combine the concept of a security kernel with hardware security features of modern mobile phone processors. On top of this layer, we use isolated execution compartments to separate applications that process medical data (e.g., an EHR viewer) and applications that process non-medical data (e.g., the telephony application or an ordinary web browser).

As a secure authentication tool, we propose a *trusted wallet* that protects the user's login credentials and performs the authentication to e-health (or other) servers on behalf of the user. This tool protects the users from being tricked into entering their credentials in malicious applications or faked web sites, and takes advantage of the underlying security framework to protect the credentials from malicious software potentially running on the phone. We present a new implementation of this wallet for mobile phones based on the Nokia N900 platform.

Compared to commodity mobile phone operating systems, our approach provides a secure environment against software attacks like malware. The usage of security-critical data like patients health records is effectively isolated from other software running on the phone, and secret data like login credentials to healthcare information systems is protected by advanced hardware security features.

In the following, we describe the usage and adversary scenario we consider (Section 2). Then, we present our security architecture (Section 3): first from a generic perspective, which can be used on all platforms, followed by its instantiation on mobile phone platforms. In Section 4, we describe how our architecture can be implemented and we present our Mobile TruWallet prototype. Finally, we conclude in Section 6.

2 Problem Scenario

We consider a scenario in which electronic health records (EHRs) of patients are stored on a local server of a healthcare provider, e.g., in a hospital. Health care professionals, like physicians and nurses, are equipped with mobile computing devices (smartphones) on which they can create, edit, and view EHRs. The EHRs are stored on the e-health server, and the smartphones communicate with the server via wireless network connections. For instance, the access of medical data can be realized with web-based applications, using standard web browser software on mobile devices. Figure 1 depicts the scenario we consider.

Since EHRs are very security-sensitive private data, and in most countries protected under strong privacy laws, unauthorized access to these data must be prevented. An adversary may try to eavesdrop or manipulate the sensitive data. As mentioned before, end-user devices are typically the least specified and least secured devices in healthcare infrastructures. Hence, an adversary would most likely try to attack the mobile phone and its communication connection to the server in order to illegitimately access medical data.

Studies like [40] have analyzed how to secure the data transfer, i.e., via encryption (for confidentiality), digital signatures (for integrity and authenticity),

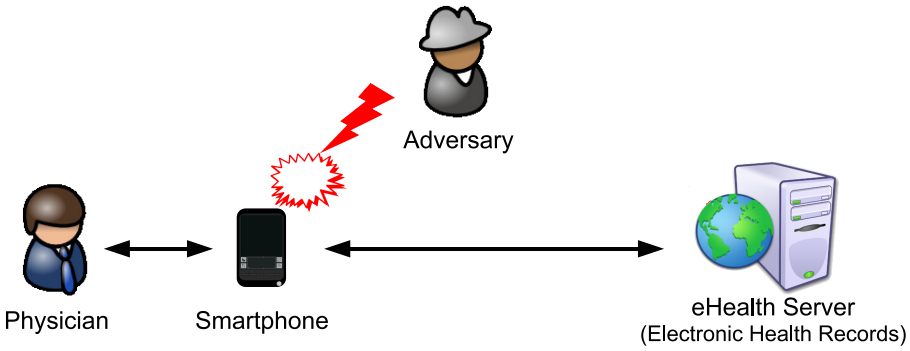

Figure 1. Use Case and Adversary Model

and user authentication (for legitimacy of access). However, the protection of the critical cryptographic keys that are needed for those mechanisms is not addressed appropriately. Hence, an attacker who gains access to these keys can circumvent any other protection mechanism.

Therefore, in this paper we concentrate on an adversary model in which the attacker targets the mobile computing device of health care professionals in order to obtain the secret login credentials or keys that are needed to access the EHR server. Once the adversary has access to these credentials, he can download or modify all medical data from the server to which the credentials allow access to. To achieve this goal, the adversary can follow two strategies:

- *Direct Access*: The adversary tries to directly access the sensitive data or keys. He could try to manipulate software running on the phone to access the data, or he could steal the device and try to access the data. The former could be achieved by letting the users install malicious software (malware, such as trojan horses) without their notice, e.g., when they browse to a website containing malicious code that exploits a vulnerability of the phone's software to install the malware. Physicians may use their phones also for other tasks and they may want to download additional applications to run them on the phone, which could create the vulnerability for such an attack.
- *Indirect Access*: The adversary tries to trick the users to enter their passwords into a faked EHR viewer application. The faked application looks like the real one, but instead of logging into the server, it sends the passwords to the adversary. The faked application could be installed on the phone in the same way as malware described above.

The problem with a commodity mobile phone operating system (OS) is that it cannot provide a sufficient level of protection for the applications or stored credentials. A mobile phone OS that is directly derived from a desktop OS (e.g., Linux or Windows) has limited protection capabilities, i.e., simple process isolation and access control. However, malicious applications can modify or eavesdrop

data of other applications since they are running with the same user privileges as other applications.

A more advanced OS, e.g., like SELinux [25], can enforce mandatory access rules, which provide a stronger isolation of different applications from each other. For instance, a text editor could only edit text files, whereas an audio application could not modify text files. The application of such a system in a mobile e-health scenario has been shown earlier [2]. However, SELinux is a very complex system with security policies that are hard to configure correctly even for moderately complicated scenarios. Moreover, due to a relatively large code base, it is infeasible to perform a comprehensive formal (or even semi-formal) verification of the correctness and security of SELinux. Another example is Android [16], which provides a similar application-oriented access control, i.e., it defines for each application different access rules and privileges — in contrast to user-oriented access control as in normal Linux and Windows, where all programs of one user share the same access rights.

Nevertheless, even advanced mobile phone OS's still suffer from ineffective protection against unauthorized modifications of programs or even modifications of the OS itself. An adversary could install on the user's phone additional (faked) programs or replace existing programs. The user has seldom a chance to notice the modification, and critical data like credentials could be transferred to the adversary.

3 Wallet Architecture

3.1 General Idea

Our security architecture aims to protect against the attacks described above. To counter direct access attacks, our architecture is based on a security kernel that isolates different applications, supports secure boot, and provides secure storage. Hence, authentication data is stored encrypted, and can only be accessed by the legitimate application (TruWallet) when the correct (unmodified) system has been booted.

Our wallet architecture aims to prevent indirect attacks by letting the wallet handle all authentication procedures. During a normal authentication, users do not enter passwords (this is automatically done by the wallet), hence they cannot accidentally disclose them towards a fake application that tries to spoof the look and feel of the legitimate EHR viewer or another application trusted by the user.

Our wallet-based security architecture provides two levels of protection (cf. Figure 2):

1. Protection of *authentication data*: TruWallet protects the user's credentials (username and password) against unauthorized access. This approach is generic, and can be used also for other scenarios (e.g., web applications like eBay or Amazon). Indeed, TruWallet can be used simultaneously by different applications, yet it only authenticates each application to the server where it has been registered as legitimate application before.

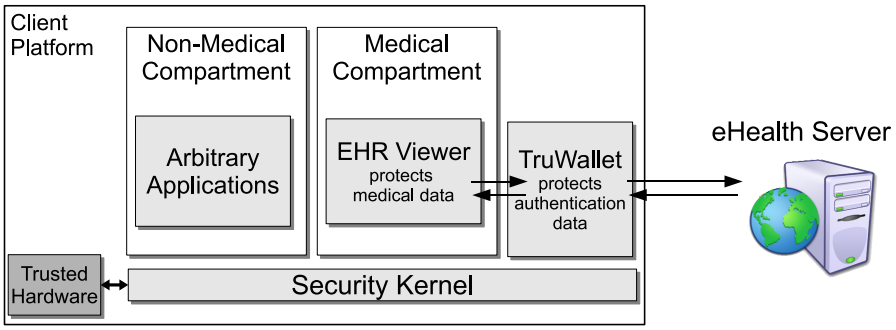

Figure 2. General idea of TruWallet

2. Protection of *medical data*: An isolated EHR viewer (which can be a special-purpose browser) is used to view EHRs. This viewer cannot be modified because a fixed program image is executed, which is measured by the security kernel by computing a cryptographic hash and compared to a known-good reference value. This ensures that all modifications of the EHR viewer can be detected. In case a browser is used as EHR viewer, this browser is only allowed to contact the EHR server and cannot connect to other sites. For all other websites or services, a separate browser process would be used which is isolated from the EHR viewer.

3.2 System model

Our system model for TruWallet consists of several parties (see Figure 3): A *user* interacts with a computing platform through a secure graphical user interface *secure GUI*. An *EHR viewer* is used to render content that it gets from the *wallet*, which is acting as a proxy. The wallet obtains the requested content from the server, blinds security-sensitive fields (e.g., password) on the pages presented to the browser, and fills in login credentials when logging into the system. For this, TruWallet has to handle two different SSL sessions: one between wallet and EHR viewer, and one between wallet and server. The secure GUI controls the input/output devices and multiplexes the screen output of the EHR viewer and of the wallet. Moreover, it always indicates the name of the application the user is currently interacting with via a reserved area on the screen, hence providing a *trusted path* between user and application. Moreover, our architecture includes a compartment for non-medical data and applications. This compartment is strictly separated from the EHR viewer and can be used for arbitrary applications.

3.3 Generic Wallet-Based e-Health Architecture

The generic TruWallet architecture is based on a *security kernel*, which is a small trusted software layer, providing *trusted services* and *isolated compartments*.

Thus, the security kernel ensures runtime security of the system. Compartments contain arbitrary software, e.g., a complete legacy operating system (Linux in our case), and may communicate only via well-defined interfaces. In particular, a malicious compartment cannot read arbitrary memory of other compartments. In our solution, EHR viewer, non-medical applications and wallet run in different compartments, and we assume that arbitrary software (including malware like Trojan horses and viruses) may be running in the non-medical compartment. Therefore, the security of our solution is based on trusted components (wallet and EHR viewer) that are executed in separated compartments, isolated from untrusted software that might be running simultaneously on the same platform.

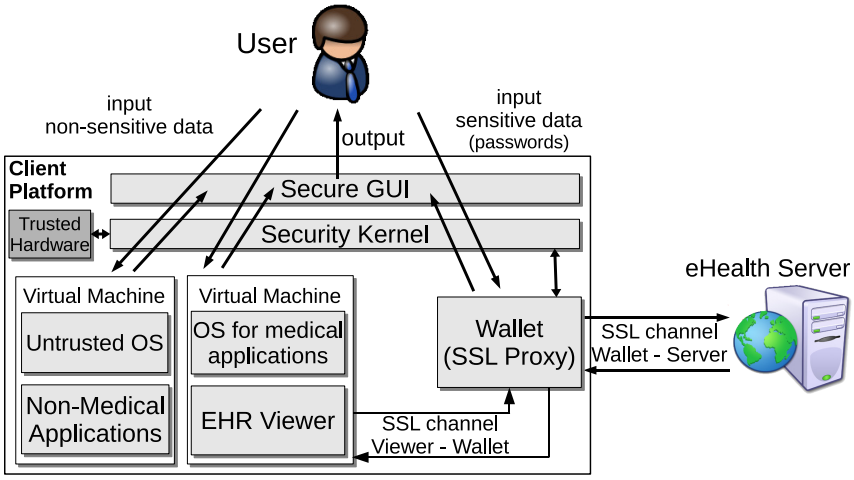

Figure 3. Generic TruWallet Architecture

In an earlier work [14], we have demonstrated the feasibility of the wallet architecture on a PC platform. In the PC-based implementation, the compartmentalization was realized by using the isolation property of virtual machines combined with the resource sharing control of an underlying microkernel. The wallet compartment is trusted, which is motivated by the fact that the complexity of the wallet is much lower than that of an EHR viewer or a compartment containing several different applications. Moreover, users cannot install arbitrary software (which may be malicious or flawed) in the wallet compartment, but they may install arbitrary viewers or other tools into other compartments. To prevent unauthorized access by other users to the platform and, hence, the sensitive data, the security kernel requires an overall user authentication (e.g., a user password) to login into the whole system. In this way, the credentials stored by the wallet are bound to the corresponding user.[2]

[2] In fact the security kernel has to provide comprehensive user access control as in typical operating systems, including system login and screen lock functionality, in

Trusted Computing support. Trusted Computing (TC) hardware and TC-enabled software is used to provide *authenticated boot*, i.e., based on a "chain of trust", the integrity of the software stack including the Trusted Computing Base (TCB) can be verified later, e.g., before access to cryptographic keys is allowed. An alternative to authenticated boot which is usually used on for mobile platforms is *secure boot*: In this case, the system's integrity state is compared to reference values and can be started only if it has not been modified.[3] Moreover, TC hardware can be used for secure storage, i.e., encryption keys protected by the hardware can only be used if load-time integrity of the system is maintained. Thus, the credentials stored by the wallet are bound to the TCB to prevent an adversary from gaining access to the data by replacing software (e.g., booting a different OS). On the PC platform [14] we used a Trusted Platform Module (TPM) [38] as TC hardware for TruWallet. The TPM is a dedicated security chip that provides – among other features – cryptographic operations (e.g., key generation, encryption, digital signatures), support for authenticated boot, and the possibility to bind cryptographic keys to the load-time integrity state of the system.

3.4 Mobile TruWallet

To implement our security architecture for mobile e-health scenarios, several building blocks for mobile environments are required:

- Trusted hardware for mobile platforms which supports features to protect cryptographic keys and verify the system integrity;
- a secure hypervisor layer for mobile platforms to provide isolated execution environments for applications;
- a security kernel with a secure GUI for mobile platforms to provide a trusted path between the user and applications, and with secure storage for applications;
- a trusted wallet (TruWallet) to handle authentication and protect the user's credentials.

In the following, we briefly introduce the first three building blocks, before we focus in more detail on the implementation of a trusted wallet on a mobile phone in the next section.

Trusted hardware for mobile platforms. The architecture of TruWallet relies of trusted hardware for performing security critical operations. To instantiate TruWallet architecture on a mobile phone, we have to use mobile hardware security extensions instead of a TPM (which is not available on current phones). On mobile platforms, general-purpose secure hardware such as M-Shield [8] and

order to prevent unauthorized access to the wallet. However, the details of those mechanisms are out of scope of this paper.

[3] Of course, it is important that these reference values are stored in a secure location, e.g., protected by security hardware, to avoid manipulations.

TrustZone [4] is available. In this paper, we focus on M-Shield, because this hardware extension is available in some current mobile phones, including Nokia N900 (which we used for our prototype).

M-Shield provides a small amount of dedicated on-chip ROM and RAM as well as one-time programmable memory for device keys which are accessible only in a special execution mode of the main CPU – the Trusted Execution Environment (TrEE). A secure state machine (SSM) guarantees secure switching between both processor modes, thus the TrEE and normal execution environment are isolated from each other. M-Shield enables the TrEE on a device with the following features: (i) isolated secure code execution; (ii) secure boot; (iii) hardware-based secure storage.

Secure hypervisor for mobile devices. Several microkernels for mobile and embedded devices have been implemented, for instance the commercially available L4 microkernels OKL4 [29] and PikeOS P4 [10]. These microkernels provide isolation between user space applications, just like their counterparts on other platforms (e.g., on PCs). Therefore, they can be used for a secure hypervisor layer for a security kernel on mobile phones. In particular, the seL4 microkernel has been formally verified for correctness [21], hence taking an important step towards building a formally verifiable security kernel on top of a microkernel.

Security kernel with secure GUI for mobile devices. Besides the isolation support, a security kernel usually has to include other security services as well. One prominent – and in our case needed – service is that of a secure graphical user interface system [31]. The main properties of a secure GUI system are (i) to protect user input/output from eavesdropping or manipulation by unauthorized applications, and (ii) to provide the user a trusted path that indicates with which application the user is currently interacting.

4 Wallet Prototype on Nokia N900

In order to demonstrate the feasibility of running a trusted wallet on a mobile phone, we have implemented Mobile TruWallet, a mobile version of trusted wallet, on a Nokia N900 device.

4.1 Mobile TruWallet Architecture

Instead of implementing a security kernel on a mobile device – for which we refer the reader to [10,21,32] – we used Maemo [27] as the basis for our mobile wallet. Maemo is a Linux-based operating system that provides standard process isolation and discretionary access control. Though Maemo does not provide the same security properties as a security kernel, we think it serves sufficiently to demonstrate the concepts of our approach. In a real product deployment, a security kernel implementation would be used instead.

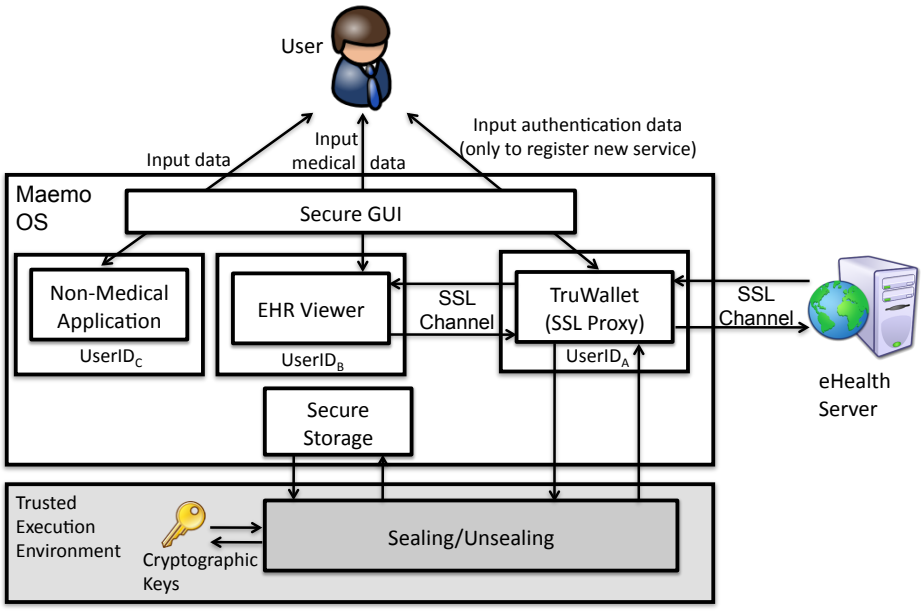

Figure 4. Mobile TruWallet Architecture

The architecture of Mobile TruWallet we have implemented is depicted in Figure 4. As the figure shows, TruWallet resides on the operating system side, but also operates on secrets at the same time, e.g., maintains a TLS channel to the web-server and also performs authentication with the user passwords. However, our generic architecture assumes that TruWallet is isolated from the rest of the system. This assumption is reasonable to some extend in the context of the Maemo operating system because Maemo's security model is based on Discretionary Access Control (DAC) which enforces security by process ownership.

We achieved process isolation on Maemo by creating a Mobile TruWallet process under a unique UserID and defining restrictive access rights to that UserID. Note that for this prototype, we rely on the standard Unix/Linux discretionary access control security framework, and there is always the threat that an administrator (root account) with the super-user access rights is compromised. However, as mentioned before, we implemented the wallet as if it was running on a security kernel. This approach allows us to concentrate on the wallet-specific aspects for the prototype (i.e., performance, user interface, compatibility to the mobile web browser and web sites, etc.). In a later stage, the wallet can be easily adapted to a system like the L4-based security kernel on the N900 [32].

The Nokia N900 device is based on M-Shield secure hardware. We utilize M-Shield functionality for the secure boot, and we also implement a secure storage functionality on top of M-Shield. Note that even if an attacker could compro-

mise the operating system, all cryptographic keys are protected by the security hardware and cannot be disclosed or copied by the attacker.

Only authenticated programs, so-called protected applications (PAs), can be executed within the TrEE of M-Shield. However, protected applications have to be authorized, i.e., certified, by the device's M-Shield stakeholder, most likely the device manufacturer. For our prototype, where it is unrealistic to obtain a certificate of the PA from the device manufacturer, we use a different approach: We reuse the general purpose APIs available for the M-Shield TrEE. This approach allows third parties to leverage the TrEE. For instance, the On-board Credentials platform (ObC) [23] provides the means to develop programs for the TrEE without the involvement of the manufacturer. In our implementation, we build the secure storage functionality of Mobile TruWallet on top of ObC.

4.2 ObC Architecture

Figure 5 illustrates the ObC architecture. The core component of the ObC platform – which resides in the dedicated RAM and can be executed in secure environment – is an *interpreter*. The interpreter provides a virtualized environment where "credential programs", i.e., scripts developed by third parties, can be executed. The credential programs are written using (a subset of) Lua scripting language [26] or in assembler. When a credential program is executed, the interpreter isolates it from secrets that are stored within the TrEE and from the execution of other credential programs.

The interpreter makes use of a Crypto Library which provides an interface for commonly used cryptographic primitives. It provides a sealing/unsealing function for ObC programs, which can be used to protect secret data stored persistently outside the secure environment. Sealed data is encrypted with a key which is protected by the TrEE, and the ObC platform controls the usage of this key. A device-specific symmetric key called ObC platform key (OPK) is used for the sealing/unsealing functionality.

Credential Manager is a mediator between OS side applications and components that reside within the TrEE. It provides an API for third-party developed applications. Using the API the applications can execute credential programs, and create and use new asymmetric keys. The credential manager maintains a database, in which credentials and credential programs are stored in a cryptographically sealed way.

A more detailed description of the ObC architecture can be found in [23,22].

4.3 Implementation

In our prototype, the wallet is implemented in the C programming language, contains about 2600 lines of code, and runs as separate process with a unique UserID. For the SSL/TLS proxy, we use Paros [30], which is an open-source implementation in Java, and it executes as a process with the same UserID as the wallet. We define restrictive access rights on this UserID so that other processes cannot access the data or code of the wallet.

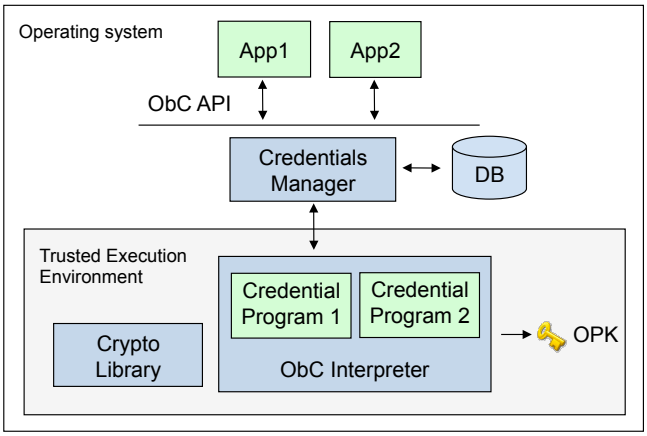

Figure 5. On-board Credential architecture

Accessing Health Records. The wallet uses the `libxml` library to parse web sites and web forms in order to search for password fields. Whenever it finds such fields, these forms are put into a cache and are disabled before they are shown in the web browser. This prevents the user from accidentally typing passwords into a potentially malicious or faked web site. Instead users just provide their user name and simply click the submit or login button in the mobile web browser.

Hence, when physicians want to access a health record from the e-health server, they simply open the EHR viewer browser on the phone and click the login button. The wallet replaces then automatically the disabled password field with the actual password of the physician's account on the e-health server. This process runs transparently, so the physician just sees the EHR viewer application, and when the login is completed he can access the health records on the server.

Registration. Before physicians can use the wallet to login to websites like the e-health server, they have to register their account in the wallet on the phone. Therefore, the wallet also looks for registration forms. When the user tries to access a website with the browser for the first time, the wallet asks the user for an existing password or it can create a new one.

Once the password has been provided (or newly created), the wallet stores the credentials in a specific file. During runtime, the access to this file is only granted to the UserID of the wallet. Hence, other programs cannot read or modify the stored credentials. When the device is going to be shut off, this file is sealed using the ObC platform as mentioned before.

Users can view a list of the stored accounts in wallet. We realized the user interface based on the Hildon GUI framework [18] on Maemo.

Interoperability. We have tested our wallet implementation with several public websites, like web e-mail services, eBay, Amazon, etc. Registration and login

work transparently and without noticeable performance overhead for the user. Hence, it should be easy to integrate web-based e-health services on our platform. Special-purpose EHR viewers or other medical applications can be supported as well as long as they use SSL/TLS and web-based login procedures. Other authentication protocols could also be integrated, but may require some effort to adapt the wallet.

5 Related Work

Previous works on secure operating systems, e.g., [13,20,33], have shown how to achieve strong isolation for secure execution and to have less complexity for the trusted computing base, i.e., the code that all security relies upon. The concept of a *security kernel* [5] incorporates all relevant functionality needed to enforce the security into a kernel that is isolated and protected from tampering by other software and small enough to be verifiable for its correctness and security. While earlier systems suffered mostly from poor performance in those days, recent CPU hardware technology, especially their virtualization support, and the development of efficient microkernel software architectures [24] allow for the realization of security kernels with low performance overhead while maintaining compatibility to existing applications. For example, Turaya [12] and the OpenTC security architecture [36] are research efforts that take advantage of these technologies to develop a security kernel on modern CPU hardware.

The Turaya Trusted Mobile Desktop [32] implements a security kernel with a secure user interface for mobile devices. Its TCB consists of a hypervisor layer and a trusted software layer. The hypervisor layer is implemented on top of an L4 microkernel, which has been ported to the Nokia N900 mobile phone. The Trusted Software Layer contains a number of security services, such as a secure graphical user interface (called TrustedGUI), a virtual private network (VPN) client, and a file encryption service. An implementation such as the Turaya Trusted Mobile Desktop can be used as an underlying security kernel for our architecture.

The protection of health records on smartphones has been addressed in other works [3,15]. However, the focus of these works is the encryption of the health records and storing the encrypted records directly on the mobile device. This approach is orthogonal to ours, as we do not consider to store the health records on the phone, but rather to protect the viewing and the access to them. In both cases, health records have to be shown in plaintext on the device at some point of time. Our architecture ensures their runtime protection by executing the EHR viewer in a separate application environment. In addition, our approach protects the credentials by leveraging trusted hardware functionality, whereas the approaches of [3,15] employ a software-only solution.

6 Conclusion and Future Work

Mobile access to electronic medical data is an emerging application scenario with strong security and privacy requirements that is rapidly gaining practical importance. Existing systems suffer from a lack of appropriate protection for security- and privacy-critical data and applications. Moreover, standard operating systems do not use existing hardware security features of mobile platforms to their full extent.

To enable secure mobile access to electronic health records containing privacy-sensitive data, we propose an e-health security architecture which protects the user's authentication credentials as well as the sensitive medical data. Our architecture is based on commonly available trusted hardware components, a security kernel, and a trusted wallet. In this paper, we introduce our comprehensive security architecture, discuss security building blocks on mobile phones, and present our implementation of Mobile TruWallet on a commodity smartphone.

Since our Mobile TruWallet prototype demonstrates the feasibility of the architecture on mobile phones, future work includes the integration of all security building blocks (i.e., the use of hardware security features, a security kernel consisting of a secure hypervisor and a trusted software layer, and the Mobile TruWallet authentication solution) into one system.

ACKNOWLEDGMENTS

This work was partially funded by the German federal state North Rhine-Westphalia and supported by the European Regional Development Fund under the project RUBTrust/MediTrust. Further, the author Alexandra Dmitrienko was supported by the Erasmus Mundus External Co-operation Window Programme of the European Union.

References

1. Mayank Aggarwal and Troy Vennon. Study of BlackBerry proof-of-concept malicious applications. Technical Report White paper, SMobile Global Threat Center, Jan 2010.
2. B. Agreiter, M. Alam, M. Hafner, J. P. Seifert, and X. Zhang. Model driven configuration of secure operating systems for mobile applications in healthcare. In *Proceedings of the 1st International Workshop on Mode-Based Trustworthy Health Information Systems*, 2007.
3. Joseph A. Akinyele, Christoph U. Lehmann, Matthew D. Green, Matthew W. Pagano, Zachary N. J. Peterson, and Aviel D. Rubin. Self-protecting electronic medical records using attribute-based encryption. Cryptology ePrint Archive, Report 2010/565, 2010. `http://eprint.iacr.org/2010/565`.
4. Tiago Alves and Don Felton. TrustZone: Integrated hardware and software security. Technical report, ARM, July 2004.
5. J.P. Anderson. Computer security technology planning study. Technical Report ESD-TR-73-51, AFSC, Hanscom AFB, Bedford, MA, October 1972. AD-758 206, ESD/AFSC.

6. Android Open Source Project. Project website. http://www.android.com, 2010.
7. Apple Inc. iOS website. http://www.apple.com/iphone/ios4, 2010.
8. Jerome Azema and Gilles Fayad. M-Shield^{TM} mobile security technology: making wireless secure. Texas Instruments White Paper, February 2008. http://focus.ti.com/pdfs/wtbu/ti_mshield_whitepaper.pdf.
9. Giuliano Benelli and Alessandro Pozzebon. Near field communication and health: Turning a mobile phone into an interactive multipurpose assistant in healthcare scenarios. In *Biomedical Engineering Systems and Technologies, International Joint Conference, BIOSTEC 2009, Revised Selected Papers*, volume 52 of *Communications in Computer and Information Science*, pages 356–368. Springer, 2010.
10. J. Brygier, R. Fuchsen, and H. Blasum. PikeOS: Safe and secure virtualization in a separation microkernel. Technical report, Sysgo, September 2009.
11. Alexandra Dmitrienko, Zecir Hadzic, Hans Löhr, Ahmad-Reza Sadeghi, and Marcel Winandy. A security architecture for accessing health records on mobile phones. In *Proceedings of the 4th International Conference on Health Informatics (HEALTHINF 2011)*, pages 87–96. SciTePress, 2011.
12. EMSCB Project Consortium. The European Multilaterally Secure Computing Base (EMSCB) project. http://www.emscb.org, 2005–2008.
13. L. Fraim. SCOMP: A solution to the multilevel security problem. In *IEEE Computer*, pages 26–34, July 1983.
14. Sebastian Gajek, Hans Löhr, Ahmad-Reza Sadeghi, and Marcel Winandy. TruWallet: Trustworthy and migratable wallet-based web authentication. In *The 2009 ACM Workshop on Scalable Trusted Computing (STC'09)*, pages 19–28. ACM, 2009.
15. Ryan W. Gardner, Sujata Garera, Matthew W. Pagano, Matthew Green, and Aviel D. Rubin. Securing medical records on smart phones. In *Proceedings of the 1st ACM Workshop on Security and Privacy in Medical and Home-Care Systems*, SPIMACS '09, pages 31–40. ACM, 2009.
16. Google Android. Security and permissions. http://developer.android.com/intl/de/guide/topics/security/security.html, 2010.
17. Dongsoo Han, Sungjoon Park, and Minkyu Lee. THE-MUSS: Mobile u-health service system. In *Biomedical Engineering Systems and Technologies, International Joint Conference, BIOSTEC 2008, Revised Selected Papers*, volume 25 of *Communications in Computer and Information Science*, pages 377–389. Springer, 2008.
18. Hildon Application Framework. Project website. http://live.gnome.org/Hildon, 2010.
19. Vincenzo Iozzo and Ralf-Philipp Weinmann. Ralf-Philipp Weinmann & Vincenzo Iozzo own the iPhone at PWN2OWN. http://blog.zynamics.com/2010/03/24/ralf-philipp -weinmann-vincenzo-iozzo-own-the-iphone -at-pwn2own/, Mar 2010.
20. Paul A. Karger, Mary E. Zurko, Douglas W. Bonin, Andrew H. Mason, and Clifford E. Kahn. A VMM security kernel for the VAX architecture. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 2–19, Oakland, CA, May 1990. IEEE Computer Society, Technical Committee on Security and Privacy, IEEE Computer Society Press.
21. Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. seL4: Formal verification of an OS kernel. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles*, Big Sky, MT, USA, Oct 2009. ACM Press. To appear.

22. Kari Kostiainen, Alexandra Dmitrienko, Jan-Erik Ekberg, Ahmad-Reza Sadeghi, and N. Asokan. Key attestation from trusted execution environments. In *TRUST 2010: Proceedings of the 3rd International Conference on Trust and Trustworthy Computing*, pages 30–46. Springer, 2010.
23. Kari Kostiainen, Jan-Erik Ekberg, N. Asokan, and Aarne Rantala. On-board credentials with open provisioning. In *ASIACCS '09: Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, pages 104–115. ACM, 2009.
24. Jochen Liedtke. On microkernel construction. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP'95)*, Copper Mountain Resort, Colorado, December 1995. Appeared as ACM Operating Systems Review 29.5.
25. Peter Loscocco and Stephen Smalley. Integrating flexible support for security policies into the Linux operating system. In *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*, pages 29–42. USENIX Association, 2001.
26. Lua. Project website. `http://www.lua.org`, 2010.
27. Maemo. Project website. `http://maemo.org`, 2010.
28. Microsoft. Windows mobile website. `http://www.microsoft.com/windowsmobile`, 2010.
29. Open Kernel Labs. OKL4 project website. `http://okl4.org`, 2010.
30. Paros. Project website. `http://www.parosproxy.org`, 2010.
31. Jeffrey Picciotto and Jeremy Epstein. Trusting X: Issues in building Trusted X window systems –or– what's not trusted about X? In *14th National Computer Security Conference*, 1991.
32. Marcel Selhorst, Christian Stüble, Florian Feldmann, and Utz Gnaida. Towards a trusted mobile desktop. In *Trust and Trustworthy Computing (TRUST 2010)*, volume 6101 of *LNCS*, pages 78–94. Springer, 2010.
33. Jonathan S. Shapiro, Jonathan M. Smith, and David J. Farber. EROS: a fast capability system. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP'99)*, pages 170–185, Kiawah Island Resort, near Charleston, Sout Carolina, December 1999. Appeared as ACM Operating Systems Review 33.5.
34. Ali Sunyaev, Jan M. Leimeister, and Helmut Krcmar. Open security issues in german healthcare telematics. In *HEALTHINF 2010 - Proceedings of the 3rd International Conference on Health Informatics*, pages 187–194. INSTICC, 2010.
35. Symbian Foundation Community. Project website. `http://www.symbian.org`, 2010.
36. The OpenTC Project Consortium. The Open Trusted Computing (OpenTC) project. `http://www.opentc.net`, 2005–2009.
37. Don Felton Tiago Alves. TrustZone: Integrated Hardware and Software Security. `http://www.arm.com/pdfs/TZ%20Whitepaper.pdf`, July 2004.
38. Trusted Computing Group. *TPM Main Specification, Version 1.2 rev. 103*, July 2007. `http://www.trustedcomputinggroup.org`.
39. Troy Vennon. Android malware. A study of known and potential malware threats. Technical Report White paper, SMobile Global Threat Center, Feb 2010.
40. Demosthenes Vouyioukas, George Kambourakis, Ilias Maglogiannis, Angelos Rouskas, Constantinos Kolias, and Stefanos Gritzalis. Enabling the provision of secure web based m-health services utilizing xml based security models. *Security and Communication Networks*, 1(5):375–388, 2008.