Requirements for an Open Ecosystem for Embedded Tamper Resistant Hardware on Mobile Devices

Michael Hölzl
UAS Upper Austria,
Campus Hagenberg
Softwarepark 11
A-4232, Hagenberg
michael.hoelzl@usmile.at

René Mayrhofer
UAS Upper Austria,
Campus Hagenberg
Softwarepark 11
A-4232, Hagenberg
rene.mayrhofer@usmile.at

Michael Roland
UAS Upper Austria,
Campus Hagenberg
Softwarepark 11
A-4232, Hagenberg
michael.roland@usmile.at

ABSTRACT

Insufficient security and privacy on mobile devices have made it difficult to utilize sensitive systems like mobile banking, mobile credit cards, mobile ticketing or mobile passports. Solving these challenges in security and privacy, could result in better mobility and a higher level of confidence for the end-user services in such systems. Our approach for a higher security and privacy level on mobile devices introduces an open ecosystem for tamper resistant hardware. Big advantages of these modules are the protection against unauthorized access and the on-device cryptographic operations they can perform. In this paper, we analyse the requirements and performance restrictions of these hardware modules and present an interface concept for a tight integration of their security features.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Cryptographic controls, Information flow controls.; H.5.m [**Information Interfaces and Presentation**]: Miscellaneous.

Keywords

Tamper resistant hardware, trusted execution environment, transparent secure channel, open ecosystem, secure element

1. INTRODUCTION

Although mobile operating systems have support for authentication (PIN/pattern entry, face recognition, etc.) or other security features (on-device encryption) major problems still exist: they are either too complicated (e.g. long, complex password) or too easy to circumvent (e.g. 2D single image face authentication). In the context of mobile devices we have to be aware of special security threats. The portable device might get stolen or lost which could give unauthorized individuals the possibility to gain personal, business or other privacy and security critical data. But the device does not have to be stolen or lost to be risk of illegitimate usage. We have to be aware that a flash memory can not be trusted. Unauthorized access or data manipulation by third party applications is a major security threat for security-critical systems [2]. Mobile platforms usually use sandboxing together with permissions to prevent "apps" from unauthorized access to personal data or other application environments [10]. Users have to decide if they allow those permissions which is, according to Miller [10], not a good choice. Other research has also shown that sandboxing does not give sufficient security against exploitable vulnerabilities [3, 5].

One attempt to overcome these issues is to use *Embedded Tamper Resistant Hardware (ETRH)* to safely store and process security-critical data and applications. In comparison to current similar business solutions (e.g. ARM TrustZone), the advantage of this attempt is the higher security level that can be achieved by using a second processor. Examples for such hardware are smartcards, secure elements and trusted platform modules. They include a secure, tamper-resistant key storage and an auxiliary processor for cryptographic operations. A tight integration of those security methods on mobile devices will allow security critical (e.g. banking, ticketing) as well as non-security critical (e.g. gaming) applications to make use of the ETRH features. We aim to achieve a level of security which allows applications like mobile banking, mobile payment and electronic passports to be executed in a trusted environment based on these hardware modules.

In this demo we present our concept and prototype implementation of an interface towards an open ecosystem for Embedded Tamper Resistant Hardware on mobile devices. It will give every application the possibility to securely install, delete and manage own applets (small application) on the ETRH and take advantage of its security features. We also demonstrate example applications which make use of this implemented interface to enhance the security and privacy of the system. After all, we provide an *Open Ecosystem* for every application running on the mobile device to make use of all these features.

2. OPEN ECOSYSTEM FOR ETRH

In the past, many systems that base their security on tamper resistant hardware have been developed. Reference work as in [7, 9] uses smartcards as the basis for an authentication system. The basic concept is to store master keys and perform cryptographic operations in the tamper resistant environment to achieve a higher security against attacks by unauthorized individuals. Work of Urien and Kiennert [12]

MoMM 2013, Vienna, Austria

use secure elements (SE), an integrated circuit which comes with NFC [8], for managing shared resources between mobile phones, laptops or other mobile devices. They host an OPENID service and therefore rely on the secure data storage of this embedded hardware.

All of the related work in this field make use of the advantages of an tamper resistant hardware in applications:

- **Protection** of stored data **against unauthorized access** and **tampering**. A serial interface, which is controlled by the operating system of the hardware, is the only way to access this data [11].
- **Cryptographic operations** are performed directly **on the chip** which makes it difficult for intruders to acquire any information. Of course, there have been multiple side-channel attacks on tamper resistant devices like power-analysis [6], differential fault-analysis [1] or others. However, a well designed system might detect data penetration and, therefore, delete all data before they can be read.

Our goal is to give applications like those listed above an open ecosystem to make use of all the advantages of an ETRH on mobile devices. Third-party as well as pre-installed applications with different security requirements should get the possibility to easily access multiple variants of tamper resistant hardware on mobile devices. Such an ecosystem should give applications the possibility to install, delete and manage own applets (small applications) on the ETRH. However, the user and applications running on the device should not notice any performance constraints and be concerned with limited resources on the hardware (memory, processing speed). The open ecosystem also has to take care that the communication can not be interfered with and that applets can not be accessed by unauthorized individuals.

To conclude, in our terms an open ecosystem for ETRH on mobile devices has to fulfil following tasks and requirements:

1. Install, delete and manage own applets on a tamper resistant hardware. These tasks are currently performed by a closed ecosystem with one secure channel to a server back-end.
2. Make security features, like managing passwords and performing cryptographic operations on the ETRH available to all mobile device applications.
3. Communication should only be possible with own applets and no details about other applets should be accessible.
4. Information about an existing secure channel should not compromise other applet-to-application communication. In the current standard the communication only uses one secure channel to the secure element which would possibly compromise communication if multiple entities have access.
5. Authenticity of the used interface, used hardware, running applets as well as applications using their own applets must be ensured.
6. Transparency in memory and performance. Although, the embedded hardware does have very limited resources, the application and user should not be concerned about this limitation.

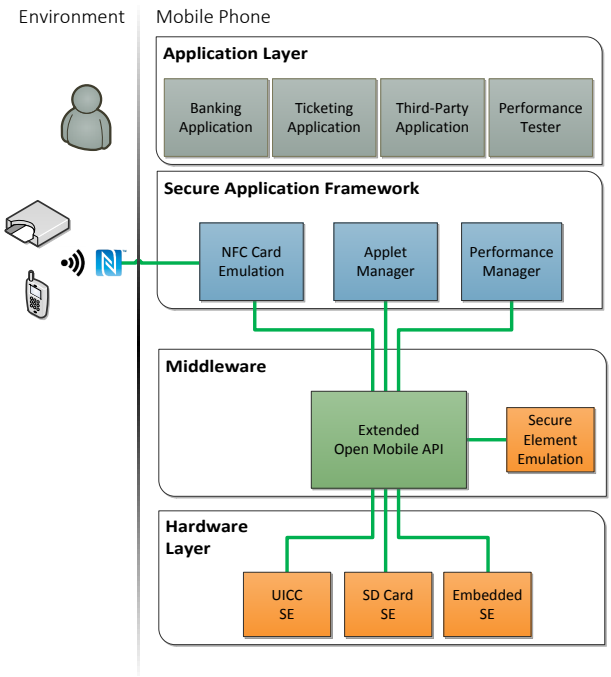

Figure 1: Architecture of the open ecosystem in a mobile phone environment.

3. CONCEPT AND PROTOTYPE

In a first step of getting towards such an open ecosystem we started with the design of an interface that allows installing, deleting and accessing applets inside a ETRH. This interface incorporates the first two requirements from section 2. The other requirements for an open ecosystem will be a matter of future work. Especially, the changes that are required to current organizational work flows are not considered in our architecture (e.g. changes required in connection to the trusted service managers (TSMs)). In this conceptual design and prototype implementation we focus only on the mobile phone as an example of a mobile device and we implement our prototypes for the Android operating system.

Figure 1 depicts the architecture of our mobile device ecosystem consisting of the four layers: applications, secure application framework, a middleware and the hardware itself. As one possible type of an ETRH we used secure elements in different variants on the mobile phone. The SE is currently available as a microSD, a UICC card or as an embedded chip in the mobile device. Our prototype implementation can communicate with all these variants and additionally comes with its own on-device emulator for easy debugging of applets.

3.1 Secure Element Middleware

Access to the hardware on mobile devices in our architecture is based on the *Open Mobile API*, a specification for an API to access secure elements.

With the middleware we also add support for secure element emulation. With this additional interface in the *Open Mobile API* it is possible to emulate an applet within the running system. The source code is installed together with the application source and runs on the application processor (in

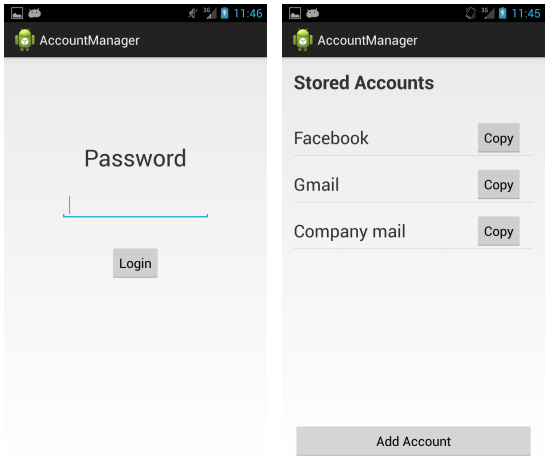

Figure 2: Account manager which stores passwords for multiple accounts on the secure element. Authentication to the applet is done by entering the master password.

Android it runs in the Dalvik VM). This makes prototyping without an actual SE very easy.

3.2 Secure Application Framework

This application framework layer represents the entry-point for the environment as well as for phone applications. It currently consists of three components: an *Applet Manager*, *Performance Manager* and an *NFC Card Emulation module* for communication with external readers and phones.

The main tasks of this framework are to manage the communication between applications and secure elements. It is responsible for setting up a secure channel and offer applications an API to install, delete and manage their own applets. The *Applet Manager* component is responsible for this management and also has to take care, together with the *Performance Manager*, that the limitations of memory and processor of the SEs are not exceeded. In the current prototype this is done by a static limit of installed applets. However, to fulfil the open ecosystem requirement of transparency in memory and performance we will conduct future research in that matter. One possible approach would be a swap-in and out procedure similar to memory management on other operating systems.

3.3 Application Layer

The components of the application layer depicted in figure 1 are only examples for the variety of use-cases we consider with our open ecosystem. Security-critical applications like mobile banking, ticketing or access control systems as well as any other third-party implementations on the marketplace should be able to use a tamper resistant hardware as basis for the security and privacy of their systems.

One application that makes use of the framework is illustrated in figure 2. The account manager stores personal passwords of web services or any other system on a specified SE. To access these accounts and passwords the user only has to remember the master key. Besides this application we implemented further prototypes which make use of the interface to access the SEs: *Applet manager*, *cryptography tester* and *performance tester*.

Table 1: Average round trip time between SD/UICC secure elements and application for different APDU cases.

	Data bytes	Case-1	Case-2	Case-3	Case-4
SD	1	19 ms	18 ms	19 ms	41 ms
	64	19 ms	109 ms	108 ms	214 ms
	128	20 ms	206 ms	200 ms	393 ms
	255	23 ms	379 ms	386 ms	776 ms
UICC	1	10 ms	10 ms	10 ms	13 ms
	64	10 ms	19 ms	18 ms	28 ms
	128	10 ms	28 ms	26 ms	41 ms
	255	10 ms	45 ms	41 ms	77 ms

4. PERFORMANCE ANALYSIS OF DIFFERENT HARDWARE VARIANTS

Opening the tamper resistant hardware platform for all applications running on the main processor raises security as well as performance concerns. Typically, this hardware comes with very limited memory and processing power. Current storage sizes of SEs vary from 10 kB to 200 kB which also restricts the amount of possibly installed applets.

To find out if the processing powers are sufficient for an open ecosystem, we performed measurements of transfer speed and cryptographic performance with SEs on mobile phones.

4.1 Transfer Speed of Hardware Variants

Communication between the framework and the SE is done with so called *Application Protocol Data Units (APDU)*. The smart card specification ISO/IEC 2816-4 [4] declares 4 different types of APDU commands:

- **Case-1:** Command header
- **Case-2:** Command header+Response data
- **Case-3:** Command header+Command data
- **Case-4:** Command header+Command data+Response data

Table 1 lists the average time of 60 round trip tests between an application and the SD/UICC card with all 4 command types. For the tests we used a DeviceFidelity CredenSE 2.8J with a NXP J3A080 chip and a Gemalto UICC, both with Java Card 2.2. The APDU exchange to the UICC is done over the Android telephony framework and the baseband. For the communication with the secure element on the SD, we have to use the standard microSD file system and write commands and responses in a temporary file on the card. The results show that the communication over the microSD file system is much slower than over the baseband channel. With a round trip time of 776 milliseconds for 256 data bytes, we would only have a data rate of 329 B/s. The UICC reaches a data rate of 3,31 kB/s for case-4 commands. For case-2 and case-3 commands (only sending or receiving data) the transfer time is nearly half the number. Due to the lack of access to the embedded SE, we were not able to perform measurements on the third possible variant.

Table 2: Average computation times of cryptographic operations on secure element

	Key length	Data	Encr.	Decr.
AES	128 bits	128 B	51 ms	54 ms
	256 bits	128 B	59 ms	62 ms
RSA	1024	117 B	53 ms	117 ms
3-DES	192	128 B	56 ms	54 ms

	Operation	Time
SHA-256	Hash of 128 bytes	78 ms
ECDH-192	Key-pair generation	76 ms
	Generate SS	103 ms
	Total	196 ms
RSA-1024	Key-pair generation	1,957 ms

4.2 Cryptographic Operations

The secure element comes with a co-processor that can perform multiple cryptographic operations in hardware. One of the major goal of the open ecosystem is to use this feature to establish a secure channel between the application and the applet. To make a feasible statement on the maximum transfer rate in such a secure channel, we measured not only the transfer speed but also the performance limitations of those cryptographic operations on the actual hardware.

Table 2 lists the result of the computation measurements for AES, RSA, 3-DES, SHA-256 and ECDH on the chip. The fastest algorithm for encryption and decryption of 128 bytes of data is AES-128. However, the other two operations gave similar encryption time for the selected key lengths. From the results we can also see that RSA key generation is significantly slower and ECDH is therefore the better choice for asymmetric cryptography. In terms of speed we can state that the secure element is able to encrypt at around 2.51 kB/s with AES-128 and at around 2.169 kB/s with AES-256 and decrypt at 2.37 kB/s and 2.06 kB/s respectively.

5. DISCUSSION

The performed measurements show that there is a significant difference in the transfer speed of the hardware variants. Standard cryptographic operations on the SE are, however, very fast due to the hardware implementation. If we combine the values, we can make a statement on the limitations of an open ecosystem when it comes to user interactions: Assuming for the secure channel between application and applet we use AES-128 to ensure the privacy of the communication. In a use-case where the user wants to encrypt or decrypt data on the SE, we would need a total time of 878 ms (SD) or 179 ms (UICC) for 255 bytes of data (transfer speed + 2 ∗ 128 bytes AES encryption). We also assume the user is willing to wait a maximum time of one second for the execution of an operation. The maximum data that could be used for such an open ecosystem without annoying the user is around 290 bytes for SD and 1,424 bytes UICC.

While this data rate might be a limitation for applications with high speed requirements, we can assume that the performance would be sufficient for applications like an account manager, a password storage or an access control system.

6. CONCLUSION

In this paper and in our demonstration we present a concept and prototype implementation of our open ecosystem with multiple example applications like a password manager, applet manager, cryptography tester and a performance tester. We will also demonstrate our prototype of a secure element emulator and the transparent access to all possible SE variants on an Android phone.

Future work will consist of further investigation on requirements for an open ecosystem of embedded tamper resistant hardware. We will especially focus on finding solutions to all tasks listed in section 2 to achieve our main goal of setting up a trusted environment for security-critical as well as other applications on mobile devices.

7. ACKNOWLEDGMENTS

This work has been carried out within the scope of u'smile, the Josef Ressel Center for User-Friendly Secure Mobile Environments. We gratefully acknowledge funding and support by the Christian Doppler Gesellschaft, A1 Telekom Austria AG, Drei-Banken-EDV GmbH, LG Nexera Business Solutions AG, and NXP Semiconductors Austria GmbH. The authors would also like to thank Endalkachew Asnake for the implementation of the transfer and cryptography tester.

8. REFERENCES

[1] R. Anderson and M. Kuhn. *Low cost attacks on tamper resistant devices*, pages 125–136. 1998.

[2] N. Ben-Asher, N. Kirschnick, H. Sieger, J. Meyer, A. Ben-Oved, and S. Möller. *On the need for different security methods on mobile phones*, pages 465–473. MobileHCI '11. ACM, 2011.

[3] S. Höbarth and R. Mayrhofer. A framework for on-device privilege escalation exploit execution on android. *Proceedings of IWSSI/SPMU*, 2011.

[4] ISO. *ISO/IEC-7816: Part 4: Interindustry commands for interchange*. International Organisation for Standardisation, Geneva, Switzerland, 2005.

[5] S. Khan, M. Nauman, A. Othman, and S. Musa. *How secure is your smartphone: An analysis of smartphone security mechanisms*, pages 76–81. 2012.

[6] P. Kocher, J. Jaffe, and B. Jun. *Differential power analysis*, pages 388–397. 1999.

[7] C.-T. Li and M.-S. Hwang. An efficient biometrics-based remote user authentication scheme using smart cards. *Journal of Network and Computer Applications*, 33(1):1–5, Jan 2010.

[8] G. Madlmayr, J. Langer, C. Kantner, and J. Scharinger. *NFC Devices: Security and Privacy*, pages 642–647. 2008.

[9] T. Mantoro and A. Milisic. *Smart card authentication for Internet applications using NFC enabled phone*, pages D13–D18. 2010.

[10] C. Miller. Mobile attacks and defense. *IEEE Security & Privacy*, 9(4):68–70, 2011.

[11] W. Rankl and W. Effing. *Smart Card Handbook*. Jun 2010.

[12] P. Urien and C. Kiennert. *A new cooperative architecture for sharing services managed by secure elements controlled by android phones with IP objects*, pages 404–409. 2012.