SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks 2014; **00**:1–15
DOI: 10.1002/sec

RESEARCH ARTICLE

An Architecture for Secure Mobile Devices

René Mayrhofer *

Josef Ressel Center for User-friendly Secure Mobile Services, University of Applied Sciences Upper Austria, Softwarepark 11, 4232 Hagenberg, Austria, rene.mayrhofer@fh-hagenberg.at

ABSTRACT

Mobile devices such as smart phones have become one of the preferred means of accessing digital services, both for consuming and creating content. Unfortunately, securing such mobile devices is inherently difficult for a number of reasons. In this article, we review recent research results, systematically analyze the technical issues of securing mobile device platforms against different threats, and discuss a resulting and currently unsolved problem: how to create an end-to-end secure channel between the digital service (e.g. a secure wallet application on an embedded smart card or an infrastructure service connected over wireless media) and the user. Although the problem has been known for years and technical approaches start appearing in products, the user interaction aspects have remained unsolved. We discuss the reasons for this difficulty and suggest potential approaches to create human-verifiable secure communication with components or services within partially untrusted devices. Copyright © 2014 John Wiley & Sons, Ltd.

KEYWORDS

mobile device security; user authentication; secure channel; virtualization; embedded smart card

Received . . .

1. INTRODUCTION

*

Mobile devices in their various physical incarnations such as smart phones, wrist watches, glasses, or other forms of wearable computing are replacing traditional clients for accessing information services. They already store highly sensitive and private data and are involved in processing monetary transactions; in the near future, they will most likely also process medical data and represent the user in even more situations by acting as their digital proxy, e.g. for digital identification.

With the transition from stationary to small mobile devices, users gain mobility, sensing capabilities, context awareness, and integration, but lose extensibility based on well-known hardware interfaces. In the past, these interfaces (e.g. USB, PCMCIA, SD, or PCI) have been used on desktops and laptops to connect trusted hardware components (e.g. smart card readers with integrated keypad, USB mass storage devices with fingerprint readers, etc.).

From a security point of view, these integrated security devices can provide trusted services such as key storage or cryptographic computations even under the assumption of a (partially) untrusted platform as the rest of the computing system. In fact, class-2 smart card readers with integrated pinpads are a required hardware component in many national laws on so-called 'advanced' electronic signatures with 'qualified' certificates [1] to prevent the well-known problem of malware logging – and invisibly using – the PIN codes that are required to unlock private keys stored

*This is the preprint version of the following article: René Mayrhofer: "An architecture for secure mobile devices", Security and Communication Networks, Wiley, 2014, which has been published in final form at http://onlinelibrary.wiley.com/doi/10.1002/sec.1028/abstract.

on users' smart cards. For mobile devices, no comparable standard solution has been adopted so far.

On the one hand, current mobile devices do not easily offer the required interfaces to connect such trusted hardware components, nor do we expect users to desire carrying a second device to use only for security relevant input or output. On the other hand, the software stack of mobile devices like smart phones or watches cannot be fully trusted and will not be able to reach sufficiently high levels of certification (such as CC EAL 4+) because of the inherent complexity of the combined set of kernel, libraries, system binaries, runtime execution environments, and installed applications that almost necessarily leads to security relevant bugs (e.g. [2, 3]). We therefore face the problem of providing comparable – or for some future scenarios even better – security on hardware/software platforms that cannot be fully trusted.

Traditionally, cryptographic protocols have been applied to secure communication over untrusted channels, e.g. by using TLS/SSL to connect to a web server over wireless channels. The problem in this case is that end users cannot directly execute cryptographic protocols without any help: they cannot directly verify digital signatures or perform a decryption operation for a modern cipher. This leads to the central open issue in mobile device security: *how can users trust their communication with an embedded secure hardware component* (e.g. a smart card embedded within their smart phone) *when they cannot realistically trust the normal user interface of their device* (which is built upon many layers of complex software components)?

In this article, we first analyze the main security threats for mobile devices (section 2) and discuss potential technical solutions to some of these threats (section 3) based on a literature survey of recent mobile security research. We then systematically analyze the corresponding usability issues and their impact on the outlined technical approaches (section 4) as a basis for describing the identified main open issue of creating an end-to-end secure channel between informational services and the end user (section 5). Finally, we propose an architecture for secure mobile devices that addresses most of the security threats while taking into account the limitations posed by usability issues (section 6). Related work is discussed throughout the article instead of condensed into one separate section. A preliminary version of this article without the in-depth literature survey and our proposed solution has previously appeared as a conference paper [4].

Our main contributions are to:

1. systematically analyze the solved and open security issues of mobile devices based on current state-of-the-art (Table I);
2. outline an approach to solve the main issue of end-to-end security with the user (section 5); and
3. propose an architecture that combines all the necessary approaches to create secure mobile "smart" devices under current conditions (section 6).

2. SECURITY THREATS FOR MOBILE DEVICES

Before discussing specific technical approaches, we analyze the security threats that current mobile devices and their users face. We illustrate the threats with the following use cases in mind: physical access control (smart phone as wireless key), mobile payment (smart phone as credit/debit card or with locally stored digital coins such as Bitcoin [5]), the smart phone as identity document (virtual passport, driving license, etc.), and the smart phone as access terminal for remote data (e.g. company email, ERP, etc.). Note that in contrast to Egners et al. [6] who distinguish between owner threats, platform threats, threats to other users, and mobile network operator threats, we focus on threats to the end user and their data (owner threats) as the most challenging class. In contrast to La Polla et al. [7], we do not assume mobile phones to be different from desktop/laptop type systems in terms of CPU and/or memory capability, but focus on usability and the context of use as the main distinguishing factors.

All of the above use cases assume the basic security requirements, namely that user data should remain confidential (C), that user data integrity (I) should be protected, and that this data and services need to remain accessible (A) to authorized users. We can easily derive that the corresponding threats are leaking private data, modifying user data, or rendering the device or its communication inaccessible. These abstract threats

Security Comm. Networks 2014; **00**:1–15 © 2014 John Wiley & Sons, Ltd.
DOI: 10.1002/sec
Prepared using secauth.cls

translate to more detailed threats on different aspects of the mobile device and its use:

Physical access: Mobile devices are small by definition, and can therefore easily be lost or stolen [8] and subsequently fall under control of others. Devices under the physical control of illegitimate third parties should still protect the private data (e.g. digital coins or the identity documents) of their legitimate owners. This is also referred to as a "malicious user" threat and is currently mostly addressed for *loss* or *theft* by on-device encryption (cf. next section) but largely open for *borrowing* of mobile devices [9]. A sub-class is the illegitimate use of devices by their owners to e.g. circumvent copy protection schemes (which falls under platform threats in the classification in [6])

Communication: As in most related work, we assume a Dolev-Yao attacker [10] on all wireless channels: an adversary can eavesdrop, delay, drop, replay, spoof, and modify messages and masquerade as any sender. Additionally, relay attacks e.g. on the embedded smart card may not directly attack the cryptographic protocol, but still be able to exploit communication [11].

Platform: Due to standard security issues in operating systems, libraries, or applications, the platform itself can be attacked with the aim of violating any of the security assumptions, e.g. to read private user data, modify data, or perform a denial-of-service attack. We can further distinguish between:

- External attacks exploiting any of the multitude of wireless interfaces (e.g. WLAN, Bluetooth, NFC, or cellular network such as UMTS) or on protocols of upper layers (e.g. HTTP, HTML parsers, etc.) may be able to directly access private data or lead to remote code execution.
- Internal attacks may be performed by installing malicious apps to either read sensitive data based on standard application permissions (granted by inattentive users or not properly enforced by the platform) or might exploit further privilege escalation issues [3] to gain full access to the platform with arbitrary permissions.

User interaction: The most difficult class of threats concerns user interaction: installed applications (malware) may try to fake the look and feel of other apps or platform components, display erroneous or fake data, capture user input with key/touch logging, and mislead or confuse the user into making wrong decisions. These threats are difficult to address because they not only involve technical, but also psychological and potentially social aspects, e.g. by exploiting peer pressure or pretended authority in social engineering attacks. We discuss the major differences to laptop/desktop systems in more detail below (see section 4).

3. TECHNICAL APPROACHES

From a technical point of view, addressing the threats to end-user data and services requires securing all involved layers of current mobile device platforms:

3.1. Hardware layer

Hardware executes all firm- and software, and therefore a minimal set of trusted hardware is required as the root of the trust chain. Although trusted platform modules (TPMs, cf. [12] for an implementation on top of the Linux kernel) have been offering secure key storage and code execution, monitoring of the boot process, and extended protocols for remote attestation (allowing a device to provide proof to a remote service that it is only executing certified software) for laptops, security issues were found [13] and they have not been widely integrated into off-the-shelf mobile phones or similar devices so far. Instead, *secure elements* (SEs) are starting to appear as part of the NFC hardware stack in the form of embedded smart cards implementing the JavaCard standard. Recent results show that the JavaCard standard could be used as the basis for an open ecosystem in which third-party applications can bundle companion code (so-called "applets") to be executed securely on the smart card [14]. Unfortunately, some of the currently used cryptographic protocols to communicate with applets on the smart card are still open to relay attacks [11], but these issues can be fixed on the protocol level and do not invalidate the approach of embedded secure elements as such. We already proposed a variation of the SRPv6 protocol that can be executed on JavaCards for efficient and provably secure communication [15] and

showed that JavaCard applets can be simulated on standard Java virtual machines for better debugging support during development [16]. With these developments, we suggest that third-party apps could significantly improve their security by bundling applets alongside the code executed on the main CPU.

SEs can be utilized for secure key storage and cryptographic operations, but cannot (at the time of this writing) support monitoring the boot process to provide a trust anchor for code executed on the main application processor (AP). Therefore, implementing secure boot of the main mobile operating system (the interface to the next layer) still requires additional hardware support to verify the boot loader code. In combination with hardware compartmentalization features such as the ARM TrustZone, SEs can be used as a basis for secure mobile devices. Note that we do not assume the mass storage (e.g. NAND flash) to be secure on the hardware layer, but that – with physical access to the device – all mass storage can be read or modified.

3.2. Platform/OS layer

Platform support includes both kernel (executing without hardware restrictions on the AP) and user space (restricted by the AP) components, which should be written with secure coding practices to prevent typical classes of code-level security vulnerabilities (such as buffer overflows, missing input validation, etc.). Additionally, the platform should isolate applications against each other using sandboxing and verify executed code based on code signing. However, realistically we always have to assume security relevant bugs in the code due to the inherent complexity of current mobile operating systems (cf. [2,3]). We therefore suggest that two specific security measures should be added on the platform layer:

- On-device storage memory encryption (as implemented e.g. by Android and iOS) is an effective safeguard against malicious user threats. Even under direct physical control, sensitive user data cannot be decrypted by attackers as long as a sufficiently long cryptographic key has been used and that key is not leaked by the platform. The interesting challenge is how to store the key; either end users need to enter the key (or a password from which the key can be derived) at bootup with the obvious trade-off between usability and security, or it needs to be stored within a secure hardware component such as an embedded smart card and unlocked with secure user authentication (see below).
- To improve the security of current sandboxing approaches, virtualization can be used to address attacks against the platform by keeping the required trusted code base as small as possible (see e.g. [17] for a brief introduction based on the older Symbian OS architecture). Only the virtualization layer – often called *hypervisor* or virtual machine manager – needs to become part of the trusted base, as the main operating system (e.g. Android) cannot realistically be assumed to be secure. ARM TrustZone supports splitting code running on the AP into trusted ("secure world") and untrusted ("normal world") parts, and starting with the Cortex-A15 generation, supports full hardware virtualization to assist existing hypervisors (e.g. Xen or KVM). With these hardware extensions, fully functional hypervisors can be implemented in around 6000 lines of code [18], making them potentially verifiable using formal methods.

  An important question concerning sandboxing and virtualization that has not yet been fully addressed is that of granularity. We suggest that application sandboxing (also extended with mandatory access control (MAC) schemes such as SELinux [19]), multiple users, zones (cf. "faces" in [20]), and virtualization of full OS instances are complementary and will be used in parallel for solving different use cases (such as phone sharing vs. solving the bring-your-own-device problem). The main issue at the time of this writing is how to visualize the currently active guest system to the end user and to enable intuitive yet secure switching mechanisms [21].

We note that both on-device encryption and sandboxing are complementary to standard malware scanning techniques based on (fuzzy) pattern matching and code execution heuristics. Although sandboxing makes it harder for malware to attack the platform or other applications, exploitation of security-relevant errors in platform code and abuse of APIs by malicious apps will remain an issue. Regular and on-demand scanning of application code is

Security Comm. Networks 2014; **00**:1–15 © 2014 John Wiley & Sons, Ltd.
DOI: 10.1002/sec
Prepared using secauth.cls

therefore beneficial for disabling known-bad application behavior. However, such malware scanning suffers from the typical problem of signature database updates, which is further complicated by the limited battery run-time on mobile devices [22].

3.3. Application layer

Apps should ideally also be implemented with secure coding guidelines to avoid data leaks on the application level. However, with the assumptions of third-party app markets and low entrance barriers for app developers, we have to assume apps to be insecure and therefore require the platform to protect itself from malicious apps (see above) and focus on securing communication between apps, infrastructure services, and users (see below). For an extensive survey on previously suggested approaches for proactively or reactively detecting security breaches by malicious or faulty applications, we refer to [7].

3.4. Communication channels

Communication can be secured effectively with well-known cryptographic protocols such as TLS or IPsec to prevent eavesdropping or manipulation of user data by any parties in-transit. This includes attackers on the (wireless) transfer channels as well as malicious apps executed on the phone. With regards to confidentiality and integrity of communication, the problem is mostly solved (as long as the cryptographic primitives remain unbroken and keys are not leaked). The remaining challenge is authenticating those devices or services that are communicating with each other, which requires the user in the loop (see e.g. [23–35]) to verify bootstrapping of cryptographic protocols and to prevent man-in-the-middle attacks under the assumed Dolev-Yao attacker.

3.5. User interaction layer

User interaction involves two aspects:

- User authentication is required to ensure that the correct user (typically the owner) is interacting with the device and as additional part to safeguard against malicious user attacks. In addition to standard password/PIN entry, (static and/or dynamic) biometric features may offer a better trade-off between usability and security, but can be difficult to secure against recorded input data [36–40]. To further alleviate that trade-off, Riva et al. propose a more dynamic determination of authentication events [41] that seems a good step forwards.
- After successful authentication, the user interface itself has to be made secure so that users can reasonably assume that the data they view and enter is not leaked or modified by other application code running in parallel on the same device.

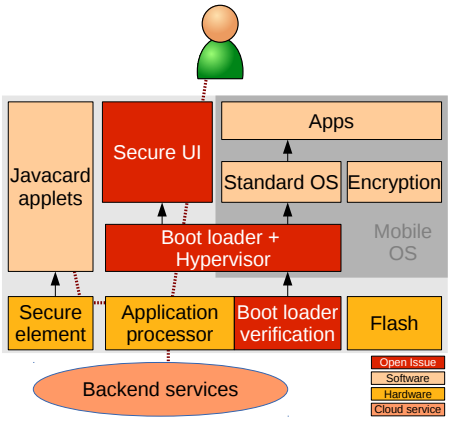

Figure 1. Overview of technical components: trust chain indicated with arrows, currently missing parts in red, end-to-end secure channel marked with dotted line

3.6. Summary of technical approaches for mobile device security

Figure 1 summarizes the existing and currently missing technical parts required to create a secure personal mobile device. The mobile operating system (OS) (e.g. Android) is fully available, but cannot realistically be assumed as sufficiently secure due to the inherent code complexity. A separate minimized operating system to drive the secure user interface (UI) can be compartmentalized from the main OS by relying on virtualization approaches based on hardware and software support (hypervisor). These two components are not yet openly available — although Mobicore aims to provide such a secure user interface based on ARM TrustZone as a hardware hypervisor, no details are openly available and, unsurprisingly, first security issues have already been discovered[†]. These technical components solve a few security threats (cf.

[†] SensePost blog: "A software level analysis of TrustZone OS and Trustlets in Samsung Galaxy Phone", 2013-06-06, last retrieved 2013-12-30 from `http://sensepost.com/blog/9114.html`.

DOI: 10.1002/sec
Prepared using secauth.cls

Table I). However, serious usability issues remain when trying to apply these approaches the same way they have been used on desktop type systems.

4. USABILITY ISSUES

Although the most prevalent classes of mobile devices currently used for accessing information services (smart phones and tablets) no longer need to be assumed to have limited resources in terms of CPU or memory capabilities (in contrast to the assumptions e.g. in [7, 42]), there are significant differences from a usability point of view that severely impact their security design:

Close personal relationship: Mobile devices are assumed to be personal devices, even more so than laptops. Especially smart phones are hardly shared with others, as they often act as a personal assistant in terms of routines of (professional and personal) daily life. This assumption creates an interesting ambivalence in terms of usable security: On the one hand, these mobile devices are often used to keep highly personal and private information, such as messages, pictures, videos, calendar entries, contacts, or location and itinerary data. On the other hand, the same assumption of being a highly personal device often leads to a puzzling neglect of security best practices, such as using long passwords for authenticating to the device or the use of on-device encryption. Many anecdotal and semi-formal user studies show that only a minority of smart phone users enables any form of regular authentication at all (and often only if forced by security policies set by administrators of some organization). Lacking hard psychological evidence, we can only assume three underlying causes for this obvious conflict:

- The assumption of a smart phone being a highly personal device carries the implication that the device is under the sole control of the respective user, because it is carried close to the body. Although this may be true most of the time, recent statistical data on mobile phone theft shows that it is a dangerous assumption from a security point of view.
- Following security best practices is bothersome (slow and error-prone) with the limited user input/output capabilities of current mobile devices.
- Users are unable or unwilling to devote explicit effort to security issues.

The latter two issues can be attributed to further differences to the laptop/desktop class of devices as discussed next.

Additionally, users may feel social pressure e.g. when lending their mobile device to a family member, friend, or colleague and may not explicitly lock sensitive functionality on the device before handing it over [9]. The result of all these effects is that – although authentication and locking mechanisms have been implemented in nearly all mobile device during the past few years – end users will often not apply them. This strongly suggests that the methods carried over from desktop and client/server scenarios have not yet been properly adapted to the mobile domain and its peculiarities concerning usability. Biometric user authentication is only one part of a potential solution, as it does not solve the issues of seemingly wasted effort and social pressure.

Limited user interface: The user interface is – after battery runtime – the biggest remaining technical limitation of current mobile devices: touch screens with on-screen keyboards and/or small thumb keyboards are not sufficiently efficient for regularly entering long passwords. Even security conscious users will find it hard to justify the time overhead of entering passwords on these limited input methods. Security measures therefore have to cope with *user input limited in length and duration*.

Multitude of contexts of use: Mobile devices are by definition used in different locations and in different contexts. Therefore, users will rarely be able to commit their undivided attention to the use of their device, but will have to remain aware of their surroundings and focused on the real-life interactions (e.g. crossing a busy road while trying to authenticate to their smart phone to read the SMS that has just been received). For this reason, security measures need to be as unobtrusive as possible, or users will – if given the choice – simply deactivate them. Additionally, security relevant interaction will also take place in a mobile context, and eavesdropping-type attacks (referred to as shoulder surfing in the context of

Security Comm. Networks 2014; **00**:1–15 © 2014 John Wiley & Sons, Ltd.
DOI: 10.1002/sec
Prepared using secauth.cls

authentication methods) can therefore not be ruled out by assuming a secure location (such as the user's office or home). All security measures have to take the *multitude of potential contexts of use* into account.

Plug-and-play expectation without training opportunities: The proliferation of mobile devices gives a wider range of the world population access to informational services. Market analysis data suggests that we already have more smart phones accessing Internet services than all laptop/desktop type systems together. This implies that more end users with no training or previous exposure to computer systems use their mobile devices for security critical transactions. In the traditional domain of laptop/desktop systems, many users received some form of training on their first contact with computer systems, and these introductions often included a part on security best practices. With wide availability of cheap smart phones outside the traditional area of computer systems, the number of untrained end users grows significantly, and all security measures therefore have to become *intuitive*.

The implication of these differences from a security point of view might be summarized as "users don't care about security". We argue that this is not the case, but that products have not yet been able to provide a sufficient compromise between usability and security that takes into account all these differences from a usability point of view. Especially the two issues of untrained users being unaware of the implications of security measures and of limited attention to security measures are the main cause for the difficulty of establishing secure communication with end users.

Table I summarizes how the threats summarized in section 2 can be addressed by technical approaches as outlined in section 3, but highlights which usability issues have to be taken into account when applying the technical approaches: Authentication (both user and device authentication) require effort and therefore time and attention from end-users, which is aggravated by the limited user interfaces. Platform protection approaches such as MAC schemes, virtualization, and secure boot need to remain largely invisible to end users.

5. UNSOLVED PROBLEM: END-TO-END SECURE CHANNEL TO THE USER

With the technical approaches outlined above (section 3), we can create a a chain of trust to secure a device platform starting from its power-off state: under direct control of the legitimate owner and starting with secure hardware as trust anchor, a chain of signed code (boot loader, hypervisor, kernel, platform, apps) could prevent the installation and execution of malware. In combination with encrypted mass storage and secure key storage in an embedded smart card, a mobile device can implement a complete secure boot process and therefore provide a protected environment as long as all signed code is reasonably secure (apps do not have to be assumed universally secure).

Current approaches developed for the mass-market (e.g. ARM TrustZone and the Mobicore secure OS) provide a comparable technical solution and extend it with the secure user interface part executed alongside the main operating system so that the standard user interface elements (e.g. the Android UI elements) no longer have to be trusted (cf. fig 1). However, two problems remain, as we have to assume malware running alongside trusted code in the form of untrusted third-party apps executed on a partially trusted (not malicious, but potentially exploitable) OS, and we have to assume users to be not as diligent as we would like them to be from a security point of view (section 4). As summarized in Table I, the main issue of lacking user attention requires that virtualization techniques need to be coupled with a secure indicator.

5.1. Securing output

Even with a secure UI (such as Mobicore) assisted by hardware virtualization (such as ARM TrustZone) and an embedded smart card (such as an NFC secure element), nothing prevents a malicious app from trying to fake the user interface that is normally presented by the secure UI and therefore manipulating output from a presumably secure app/service to the user. The reason is that both the trusted and untrusted software components rely on the same input/output modalities – the single touch screen in the case of current smart phones. Although some solutions have been developed towards a secure GUI on desktop-type systems (see e.g. the X11 windows extensions by Feske and Helmuth [43]), mobile devices require different approaches to visualization: window managers and the

Table I. Summary: Security threats for mobile devices, technical approaches to address them, and usability issues to take into account

Layer	Threat	Technical approaches	User interaction limitations
physical	loss/theft	encryption + biometric authentication	delay + effort
	borrowing	biometric authentication + virtualization	delay + effort + social pressure
	circumvent copy protection	secure key storage on smart card	offline capability
communication	subverted communication	sensor-based device authentication	delay + effort
platform	abuse APIs	MAC + malware scanning	transparency
	exploit platform code	MAC + virtualization + secure boot	should be invisible
	attack other apps	MAC + virtualization + smart card	should be invisible
user interaction	masquerade as other app	virtualization + secure indicator	user attention
	capture user input	virtualization + secure indicator	user attention

resulting window decorations are rarely available, and running applications often use full screen modes. Under the assumption of applications with access to all parts of the screen, we therefore have to deal with malware trying to copy the look and feel of other parts of the (secure) system.

We explicitly repeat that securing output to the user is significantly harder than communicating with a backend service or another device, because users cannot realistically verify digital signatures or perform decryption operations without the help of computing devices. Therefore, we cannot rely on the standard approach of using a cryptographic protocol to secure communication through untrusted components. Furthermore – considering the usability issues discussed above – end users would not devote the required effort and attention to verifying their end of a cryptographically secured communication even if they were capable of doing so.

The most sensible solution to provide users with the required visual cue in an intuitive, unobtrusive, low-effort, yet unambiguous manner seems to be additional hardware. Various options seem suitable, from a simple RGB LED that indicates which virtual guest is currently controlling the UI to a secondary display under exclusive control of the (limited) trusted components. We call this additional hardware output component the *secure indicator*, in line with the independently proposed "secure-mode indicator" from recent related work [44]. However, the really interesting question is not the technical implementation, but standardization on the user interaction: will everybody agree on *one* standard that users can become used to and that they will actually check (unobtrusive, but noticeable for every secure interaction, intuitive, understandable, and documented for first-time users)?

5.2. Securing input

We face an equivalent problem for input from the user to a presumably secure app/service: all input is currently handled by the main OS because of its exclusive hold on the single touch screen. Secure input needs a way for the user to a) be sure that input only goes to secure components (e.g. trusted virtual guest domains) and b) to initiate a switch between different components (virtual guests) that is not subject to an app-in-the-middle attack.

While this can also be achieved with additional hardware such as dedicated pinpads hard-wired to the SE, practical experience strongly suggests that users will not want their mobile devices to become bigger "just for security purposes". A more practical approach is therefore to first secure output and then rely on the intuitive end-user assumption that those components that produce the current output will also receive the input. This solves the problem concerning the user interaction aspect by giving (secure) feedback to users which component (virtual guest) they are interacting with. However, the technical implementation still remains open for current smart phone platforms, because the touch screen driver will typically reside in the main OS and not in the limited hypervisor code. Our current approach is to move touch screen event handling into the hypervisor code and forward those input events only to the virtual guest that is currently controlling screen output.

The same approach has been taken in very recent work by Gilad, Herzberg, and Trachtenberg [44] by implementing a μTCB as a small component running in ARM TrustZone secure world mode alongside the standard operating system executed in normal world mode. This μTCB controls the touch screen input interrupts and therefore receives all input as long as it is active.

Security Comm. Networks 2014; **00**:1–15 © 2014 John Wiley & Sons, Ltd.
DOI: 10.1002/sec
Prepared using secauth.cls

We consider it one possible implementation of our more abstract hypervisor component in the architecture described next. Notably, the μTCB also assumes additional input hardware in the form of a single "Secure Attention Key" to trigger a switch to the secure world code. Additional hardware is one option considered in our previous comparison of switching mechanisms from a user point of view [21]. However, preliminary results suggest that users find switching via the standard lock screen more intuitive than using a dedicated hardware switch.

6. AN ARCHITECTURE FOR SECURE MOBILE DEVICES

Taking into account all of the above, we propose an architecture for secure mobile devices based on current state-of-the-art research and off-the-shelf hardware and software components as shown in figure 2 (critical path shown with bold lines). In addition to the existing hardware components of flash memory, main CPU (application processor, AP), wireless radios (baseband processor, BP), smart card (secure element, SE), and touch screen, we argue that the following software parts need to be added or extended:

- The central component is *virtualization*, which supports executing a minimal, secure operating system alongside one or multiple standard mobile operating systems such as Android. Although software-only virtualization solutions (e.g. Xen) exist, most virtual machine managers (hypervisors) require adequate hardware support and therefore a sufficiently new CPU. In addition to managing access to memory and computation, the hypervisor will also need to control access to wireless radios and the touch screen.
- As described above, the touch screen is not sufficient to indicate to the end user which of the virtual guests is currently active. We suggest a *secure indicator*, e.g. in the form of an RGB LED that is under direct control of the hypervisor and cannot be manipulated by any of the operating systems.
- To verify that the correct virtual machine manager is controlling the main CPU, radios, and touch screen, bootup needs to be verified. The embedded smart card is the obvious control point for such a *secure boot* process, as it is the only trustworthy hardware component when a device is turned on. Ideally, the secure indicator should be directly connected to the smard card to give the smart card an opportunity to verify code executed on the main CPU during each switch to another security zone (another virtual guest).
- A *secure operating system* has the main task to provide trustworthy user interaction via the touch screen (as long as the secure indicator shows that the secure OS is currently being executed), and may act as a communication relay between applets and backend services. We suggest that, although such a secure OS should be as small as possible to minimize the attack surface, high-level languages with run-time checks should still be used to minimize typical buffer overflow and comparable coding errors in complex user interface systems.
- An *applet manager* is required to support the installation and management of third-party applets on the single embedded smart card. This applet manager should (logically) communicate only with the secure OS instance (physically, the smart card will communicate with the main CPU, and therefore access has to be mediated by the hypervisor).
- Standard *encryption* of mass storage such as flash memory should be controlled by the hypervisor to support different encryption keys for different security zones (virtual guests). Ideally, these keys should be stored securely on the embedded smart card.
- Mandatory access control (*MAC*) should be employed in each of the standard operating system instances to better restrict third-party apps and further minimize the attack surface. However, we note that this is only a defense-in-depth measure, but that we strongly suggest against relying on this measure for the overall security of the system. MAC schemes (such as SELinux) are built on top of the standard operating system kernel (such as the Linux kernel) and are subject to all security issues in these kernels, which makes them insufficient as the main approach for hardening the system.

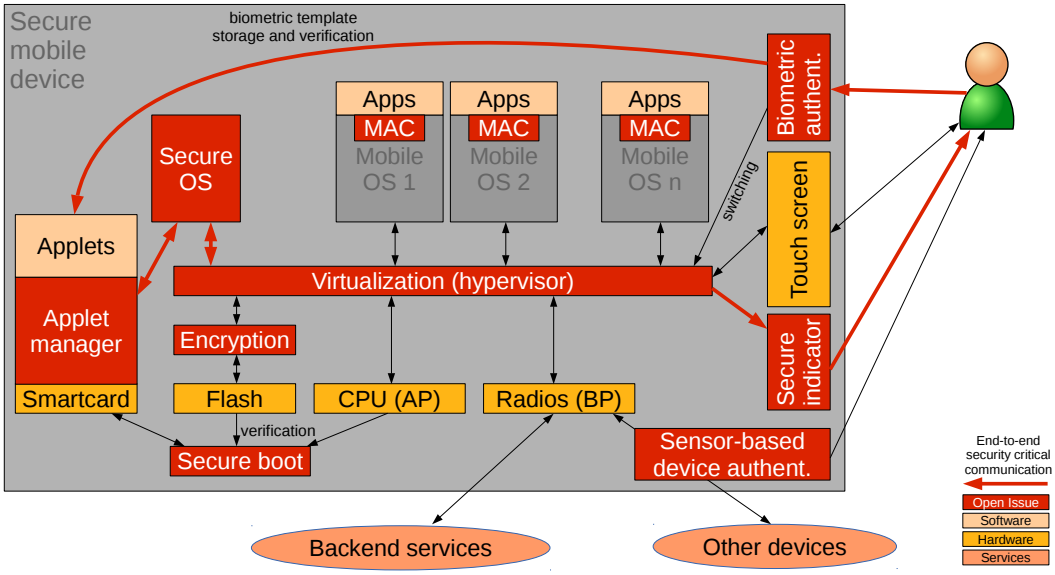

Figure 2. Proposed architecture for secure mobile devices

- *Biometric user authentication* identifies the end user. Ideally, the required biometric templates should be stored and processed in applets running on the smart card, which can then pass on the authentication events to the hypervisor for allowing or denying a switch to another security zone. Additionally, some of the operating system instances may employ additional biometric authentication schemes for more fine-grained access control within apps. However, only authentication decisions computed within the smart card or the secure OS may be considered sufficiently trustworthy.
- To authenticate communication with other devices over wireless communication channels, *sensor-based device authentication* (cf. [24]) can be used to support users in verifying which devices their own mobile device is communicating with.

This architecture addresses the identified main issue of enabling a secure communication channel with the end user. Although we cannot cryptographically secure this communication channel through untrusted components (including the touch screen and main CPU), our proposed solution only requires minimal hardware extensions such as a secure indicator and deeper integration of embedded smart cards into the boot process. In addition to the main touch screen, users can then rely on this secure indicator to show them which security context is currently active. This prevents the attack of malicious apps masquerading as the secure OS that was unsolved in previous mobile device security research. The critical path is closed by biometric authentication to safeguard against malicious user attacks.

7. PERFORMANCE CONSIDERATIONS

It is clear that any additional layer in a system architecture will have a performance impact for applications. Out of the technical approaches summarized above, we can identify five areas with measurable overhead:

Virtualization is well-studied in terms of performance overhead, and specific implementations have been optimized towards minimizing the different areas of overhead. In terms of virtualized network interfaces, recent measurements with the KVM hypervisor on an ARM Cortex-A15 CPU showed a guest system to achieve roughly 85% of the performance of the host [45] and therefore indicate a non-negligible impact for high-speed network transfers even when using the optimized “virtio” drivers for Linux host and guest operating systems. We suggest that this overhead, although significant for server side systems, is not prohibitive for mobile clients, because the wireless network links will in the majority of cases be the limiting factors. The same study shows that guests achieve around

Security Comm. Networks 2014; **00**:1–15 © 2014 John Wiley & Sons, Ltd.
DOI: 10.1002/sec
Prepared using secauth.cls

93% of the disk read performance, which is an important factor for starting applications and reading assets, but the overhead does not seem significant for standard use of mobile devices.

CPU execution and memory access speeds are sufficiently close to native speeds [45] with most operations relevant to hypervisor operation in the range of only 200-1000 processor cycles [18]. Even with intentionally simplified hypervisors such as the L4 kernel and without using hardware virtualization extensions, typical applications only suffer about 5% overhead [46].

On-device storage memory encryption incurs overhead by having to pass all read requests from the mass storage (typically a flash memory) through decryption and all write requests through encryption operations executed on the main CPU. As we are not aware of scientific literature studying the encryption overhead specifically for Android in detail, we provide benchmark results from a limited sample size as an indication of the rough overhead created by on-device encryption. On a current Asus Google Nexus 7 tablet (2013 version) with a quad-core 1.5 GHz Snapdragon S4 Pro CPU running Android version 4.4.2 and Linux kernel 3.4.0, we ran both the specialized "AndroBench" app for I/O micro- and macro-benchmarking and the widely used "AnTuTu" app for high-level application benchmarking. All tests were run 5 times in a row each before and after enabling on-device storage memory encryption to account for caching and other warm-up effects of benchmarks such as JIT optimizations. We only consider the internal memory (formatted with the ext4 filesystem and mounted as /data), but no external memory such as microSD cards due to their slower I/O busses.

As detailed in Table II, AndroBench sequential read speed decreases by over 75% and sequential write speed by about 50% for micro benchmarking cases, which is also shown in the random read and write access operations per second. Macro benchmarks reflect these overheads with between 28% (browser benchmark) and 75% (camera benchmark) slowdowns. Surprisingly, AnTuTu I/O results do not show noticeable differences with both the storage and database average slowdowns in the area of their respective standard deviations of measurements on unencrypted storage. Our conclusion is that, although AnTuTu is a popular benchmark for CPU and graphical performance, it does not represent differences in raw I/O speed well enough to distinguish between encrypted and unencrypted storage.

Mandatory access control such as SELinux demands additional policy checks for many system calls. However, I/O benchmarks with and without SELinux on an older Android device showed only few cases with statistically significant overhead caused by SELinux policy checking [19]. Mostly file meta data intensive operations such as directory listings or opening small files will see any impact, while mass data operations such as sequential reads and writes are not significantly slower. We note that network connectivity was not benchmarked, but that we do not expect performance limitations that would be significant for applications.

Secure communication channels also require additional computational effort for encryption, integrity protection (and optional compression) of mass data streams. Ignoring the computational overhead and latency of key setup during channel initialization, Qu, Li, and Dang measured the performance impact of OpenVPN on Android [47]: a Motorola Xoom tablet directly connected via local WLAN to a sufficiently fast Linux server achieved a throughput of 25.45 MBit/s and 12.84 ms latency without OpenVPN, up to 39.42 MBit/s with 16.79 ms with OpenVPN (with AES cipher) and compression, and 21.15 Mbit/s with 13.55 ms latency with OpenVPN without compression.

These results demonstrate roughly 30% increase in latency, but also 55% increase in throughput due to compression. If latency is an issue, then compression can be disabled to reduce the latency overhead to roughly 5% with 17% slowdown in throughput (presumably due to additional protocol headers in the stack). We argue that this overhead in latency as measured for the best case (local WLAN) will not be significant for wide area network connections over wireless links, and therefore the overhead of a standard SSL/TLS secure channel implementation is negligible on current mobile devices.

Key storage and cryptographic computation on smart cards allow a significantly better security level, but will be slower than on the main CPU due to the restricted resources of smart cards. We have previously shown [14]

Unencrypted	AnTuTu [rating]		AndroBench micro [MB/s, IOPS]				AndroBench macro [ms]			
	storage	database	seq.rd.	seq.wr.	rnd.rd.	rnd.wr.	browser	market	camera	camcorder
Try 1	1185	625	59,89	17,74	2526,81	288,5	82,75	214,25	203,25	488,5
Try 2	1182	630	59,92	18,29	2534,02	265,14	84,5	209,25	202	486,75
Try 3	1138	630	61,33	18,57	2479,74	290,3	86,5	205,75	195	489,25
Try 4	1012	635	63,38	20,21	2516,09	290,97	84	202,25	204	493,75
Try 4	896	630	63	19,42	2580,67	286,71	85	208,5	201,75	487,5
Mean	1082,6	630	61,50	18,85	2527,47	284,32	84,55	208	201,2	489,15
Std.dev.	125,74	3,54	1,65	0,97	36,34	10,85	1,37	4,44	3,59	2,74

Encrypted	AnTuTu [rating]		AndroBench micro [MB/s, IOPS]				AndroBench macro [ms]			
	storage	database	seq.rd.	seq.wr.	rnd.rd.	rnd.wr.	browser	market	camera	camcorder
Try 1	977	605	13,88	9,95	1504,18	203,4	113,5	275,25	287,25	628
Try 2	958	630	13,87	9,99	1497,75	203,54	105,25	271,75	275,5	808
Try 3	972	630	13,77	9,89	1502,57	200,13	109,5	442	294,75	693,25
Try 4	954	630	13,63	9,7	1499,52	143,93	102,5	309,5	454,25	924,75
Try 5	982	630	12,57	6,41	1358,1	141,52	110,25	461,5	445,5	992,5
Mean	968,6	625	13,54	9,19	1472,42	178,50	108,2	352	351,45	809,3
Std.dev.	12,12	11,18	0,55	1,56	63,96	32,70	4,34	92,50	90,16	152,67

Table II. Benchmark results for Galaxy Nexus 7 internal storage I/O, unencrypted (top) and encrypted (bottom)

that one major issue with the integration of smart cards on mobile devices is their interface to the main CPU. While a smart card in the form of a SIM card (UICC) has negligible transfer delay, microSD cards using file based data transfer can incur over 700 ms for one request/response message exchange. Data that is already on the smart card can be encrypted and decrypted fairly quickly (around 50 ms to 60 ms for AES en-/decryption of 128 bytes with 128 or 256 bit keylength, ca. 80 ms for hashing 128 bytes with SHA-256), the only notable delay occurs during RSA keypair generation (nearly 2 s).

We therefore conclude that cryptographic operations on the smart card are not the bottleneck, but that transferring data between the main CPU and the smart card is critical for performance. For production use, a fast bus connection is required, e.g. for an embedded secure element or a SIM card.

User authentication and interaction are arguably the most critical part, not only in terms of security, but also in terms of overall system performance. Authentication can consume significant time and attention from end users and therefore needs to be optimized as far as possible. We have previously worked on biometric authentication [36–40] and are currently studying the user interaction aspect of visualizing and switching between different virtual zones [21]. However, the overall issue of secure and user-friendly user authentication and interaction remains open for future research.

8. CONCLUSIONS AND OUTLOOK

After systematically identifying threats to user data on current mobile devices, we have analyzed the landscape of technical approaches to addressing them and the significant differences to traditional security measures from a security point of view. To this end, we found the need for additional technical components, namely virtualization, further integration of existing embedded smart cards with secure boot verification, and secure user interface components. The biggest open issue is how to design the user interaction in terms of data input and output in such a way that users can reliably and unobtrusively be aware of which application part they are communicating with. This is currently unsolved and remains the most important research question to advance the area of secure mobile devices. We proposed a specific architecture for secure mobile devices that builds upon existing hardware components, but extends them with multiple security measures that, in concert with each other, enable secure end-to-end communication with end users.

Security Comm. Networks 2014; **00**:1–15
DOI: 10.1002/sec
Prepared using secauth.cls

In a cooperation with other research groups, we are currently preparing an extensive user study concerning the granularity of separating different parts of mobile platforms/apps and on visualization and input concepts. Results will be published as future work and we expect them to inform our architecture decisions towards addressing the question of intuitive secure user interaction.

ACKNOWLEDGEMENTS

This work has been carried out within the scope of *u'smile*, the Josef Ressel Centre for User-Friendly Secure Mobile Environments. We gratefully acknowledge support by the Christian Doppler Gesellschaft, A1 Telekom Austria AG, Drei-Banken-EDV GmbH, LG Nexera Business Solutions AG, and NXP Semiconductors Austria GmbH. Additionally, we are thankful to the anonymous reviewer for comments that helped to improve an earlier version of this article.

REFERENCES

1. Community framework for electronic signatures. OJ L 13 of 19.1.2000 January 2000. URL `http://europa.eu/legislation_summaries/information_society/other_policies/l24118_en.htm`.
2. Egners A, Meyer U, Marschollek B. Messing with Android's permission model. *Proc. TrustCom 2012*, IEEE CS Press, 2012; 505–514, doi:10.1109/TrustCom.2012.203. URL `http://dx.doi.org/10.1109/TrustCom.2012.203`.
3. Höbarth S, Mayrhofer R. A framework for on-device privilege escalation exploit execution on Android. *Proc. IWSSI/SPMU 2011: 3rd International Workshop on Security and Privacy in Spontaneous Interaction and Mobile Phone Use, colocated with Pervasive 2011*, 2011.
4. Mayrhofer R. When users cannot verify digital signatures: On the difficulties of securing mobile devices. *Proc. HPCC 2013: 15th IEEE International Conference on High Performance Computing and Communications*, IEEE CS Press: Washington, DC, USA, 2013; 1579–1584.
5. Nakamoto S. Bitcoin: A peer-to-peer electronic cash system 2009; URL `http://www.bitcoin.org/bitcoin.pdf`.
6. Egners A, Marschollek B, Meyer U. Hackers in your pocket: A survey of smartphone security across platforms. *Technical Report AIB-2012-07*, RWTH Aachen May 2012. URL `http://itsec.rwth-aachen.de/publications/ae_hacker_in_your_pocket.pdf`.
7. Polla ML, Martinelli F, Sgandurra D. A survey on security for mobile devices. *IEEE Communications Surveys & Tutorials* February 2013; **15**:446–471, doi:10.1109/SURV.2012.013012.00028.
8. Halpert B. Mobile device security. *Proceedings of the 1st Annual Conference on Information Security Curriculum Development*, InfoSecCD '04, ACM: New York, NY, USA, 2004; 99–101, doi:10.1145/1059524.1059545. URL `http://doi.acm.org/10.1145/1059524.1059545`.
9. Karlson AK, Brush AB, Schechter S. Can I borrow your phone? understanding concerns when sharing mobile phones. *Proc. CHI 2009*, ACM Press, 2009; 1647–1650, doi:10.1145/1518701.1518953.
10. Dolev D, chih Yao AC. On the security of public key protocols. *IEEE Transactions on Information Theory* 1983; **29**:198–208, doi:10.1109/TIT.1983.1056650.
11. Roland M, Langer J, Scharinger J. Practical attack scenarios on secure element-enabled mobile devices. *Near Field Communication (NFC) Workshop 2012*, 2012; 19–24, doi:10.1109/NFC.2012.10.
12. Sailer R, Zhang X, Jaeger T, van Doorn L. Design and implementation of a TCG-based integrity measurement architecture. *Proc. USENIX 2004: 13th conference on USENIX Security Symposium*, 2004; 223–238.
13. Rudolph C. Covert identity information in direct anonymous attestation (DAA). *Proc. IFIP SEC 2007, IFIP*, vol. 232, Springer-Verlag, 2007; 443–448, doi:10.1007/978-0-387-72367-9_38.
14. Hölzl M, Mayrhofer R, Roland M. Requirements for an open ecosystem for embedded tamper resistant hardware on mobile devices. *Proc. MoMM 2013: 11th International Conference on Advances in Mobile Computing and Multimedia*, ACM Press: New York, NY, USA, 2013; 249–252.

15. Asnake E, Hölzl M, Mayrhofer R. An efficient password-authenticated secure channel for java card applets. submitted for publication 2014.
16. Roland M, Langer J, Mayrhofer R. (ab)using foreign vms: Running java card applets in non-java card virtual machines. *Proc. MoMM 2013: 11th International Conference on Advances in Mobile Computing and Multimedia*, ACM Press: New York, NY, USA, 2013; 286–292.
17. Brakensiek J, Dröge A, Botteck M, Härtig H, Lackorzynski A. Virtualization as an enabler for security in mobile devices. *Proc. IIES 2008*, ACM Press, 2008; 17–22, doi:10.1145/1435458.1435462. URL `http://doi.acm.org/10.1145/1435458.1435462`.
18. Varanasi P, Heiser G. Hardware-supported virtualization on ARM. *Proceedings of the Second Asia-Pacific Workshop on Systems*, APSys '11, ACM, 2011; 11:1–11:5, doi:10.1145/2103799.2103813. URL `http://doi.acm.org/10.1145/2103799.2103813`.
19. Shabtai A, Fledel Y, Elovici Y. Securing Android-powered mobile devices using SELinux. *IEEE Security and Privacy* 2010; **8**:36–44, doi:http://doi.ieeecomputersociety.org/10.1109/MSP.2009.144.
20. Seifert J, Conradi ADLB, Hussmann H. TreasurePhone: Context-sensitive user data protection on mobile phones. *Proc. Pervasive 2010*, *LNCS*, vol. 6030, Springer-Verlag, 2010; 130–137, doi:10.1007/978-3-642-12654-3_8.
21. Riedl P, Koller P, Mayrhofer R, Kranz M, Möller A, Koelle M. Visualizations and switching mechanisms for security zones. *Proc. MoMM 2013: 11th International Conference on Advances in Mobile Computing and Multimedia*, ACM Press: New York, NY, USA, 2013; 278–281.
22. Oberheide J, Veeraraghavan K, Cooke E, Flinn J, Jahanian F. Virtualized in-cloud security services for mobile devices. *Proceedings of the First Workshop on Virtualization in Mobile Computing*, MobiVirt '08, ACM: New York, NY, USA, 2008; 31–35, doi:10.1145/1622103.1629656. URL `http://doi.acm.org/10.1145/1622103.1629656`.
23. Kumar A, Saxena N, Tsudik G, Uzun E. Caveat emptor: A comparative study of secure device pairing methods. *Proc. PerCom2009*, 2009; 1–10.
24. Mayrhofer R, Fuss J, Ion I. UACAP: A unified auxiliary channel authentication protocol. *IEEE Transactions on Mobile Computing* April 2013; **12**(4):710–721, doi:10.1109/TMC.2012.43.
25. Mayrhofer R, Gellersen H. Shake well before use: Intuitive and secure pairing of mobile devices. *IEEE Transactions on Mobile Computing* June 2009; **8**(6):792–806. Revised and extended version of [48].
26. Groza B, Mayrhofer R. SAPHE - simple accelerometer based wireless pairing with heuristic trees. *Proc. MoMM 2012: 10th International Conference on Advances in Mobile Computing and Multimedia*, ACM Press: New York, NY, USA, 2012; 161–168.
27. Mayrhofer R. The candidate key protocol for generating secret shared keys from similar sensor data streams. *Proc. ESAS 2007: 4th European Workshop on Security and Privacy in Ad hoc and Sensor Networks*, *LNCS*, vol. 4572, Springer-Verlag: Berlin, Heidelberg, Wien, 2007; 1–15.
28. Mayrhofer R. Towards an open source toolkit for ubiquitous device authentication. *Workshops Proc. PerCom 2007: 5th IEEE International Conference on Pervasive Computing and Communications*, IEEE CS Press: Washington, DC, USA, 2007; 247–252. Track PerSec 2007: 4th IEEE International Workshop on Pervasive Computing and Communication Security.
29. Mayrhofer R, Welch M. A human-verifiable authentication protocol using visible laser light. *Proc. ARES 2007: 2nd International Conference on Availability, Reliability and Security*, IEEE CS Press: Washington, DC, USA, 2007; 1143–1147.
30. Soriente C, Tsudik G, Uzun E. HAPADEP: Human asisted pure audio device pairing. Cryptology ePrint Archive, Report 2007/093 March 2007.
31. Soriente C, Tsudik G, Uzun E. BEDA: Button-enabled device pairing. *Proc. IWSSI 2007*, 2007; 443–449.
32. Sigg S, Schuermann D. Secure communication based on ambient audio. *IEEE Transactions on Mobile Computing (TMC)* February 2013; **12**:331–340, doi:10.1109/TMC.2011.271.
33. Saxena N, Ekberg JE, Kostiainen K, Asokan N. Secure device pairing based on a visual channel. Cryptology ePrint Archive, Report 2006/050 2006.
34. Nithyanand R, Saxena N, Tsudik G, Uzun E. Groupthink: On the usability of secure group

association of wireless devices. *Proc. Pervasive 2010*, LNCS, Springer-Verlag, 2010.

35. Varshavsky A, Scannell A, LaMarca A, de Lara E. Amigo: Proximity-based authentication of mobile devices. *Proc. UbiComp 2007*, Springer-Verlag, 2007; 253–270.
36. Findling R, Mayrhofer R. Towards pan shot face unlock: Using biometric face information from different perspectives to unlock mobile devices. *International Journal of Pervasive Computing and Communications (IJPCC)* 2013; **9**:190–208, doi:10.1108/IJPCC-05-2013-0012. A preliminary version of this work was published in MoMM 2012 with a limited set of classifiers and a significantly smaller data set used for evaluation.
37. Findling R, Wenny F, Holzmann C, Mayrhofer R. Range face segmentation: Face detection and segmentation for authentication in mobile device range images. *Proc. MoMM 2013: 11th International Conference on Advances in Mobile Computing and Multimedia*, ACM Press: New York, NY, USA, 2013; 260–269.
38. Findling R, Mayrhofer R. Towards secure personal device unlock using stereo camera pan shots. *Proc. EUROCAST 2013: 14th International Conference on ComputerAided Systems Theory*, LNCS, Springer-Verlag: Berlin, Heidelberg, Wien, 2013; 417–425.
39. Muaaz M, Mayrhofer R. An analysis of different approaches to gait recognition using cell phone based accelerometer. *Proc. MoMM 2013: 11th International Conference on Advances in Mobile Computing and Multimedia*, ACM Press: New York, NY, USA, 2013.
40. Mayrhofer R, Kaiser T. Towards usable authentication on mobile phones: An evaluation of speaker and face recognition on off-the-shelf handsets. *Proc. IWSSI/SPMU 2012: 4th International Workshop on Security and Privacy in Spontaneous Interaction and Mobile Phone Use, colocated with Pervasive 2012*, 2012. Available online at `http://www.medien.ifi.lmu.de/iwssi2011/`.
41. Riva O, Qin C, Strauss K, Lymberopoulos D. Progressive authentication: deciding when to authenticate on mobile phones. *Proc. USENIX 2012*, USENIX, 2012; 301–316.
42. Oberheide J, Jahanian F. When mobile is harder than fixed (and vice versa): demystifying security challenges in mobile environments. *Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications*, ACM, 2010; 43–48.
43. Feske N, Helmuth C. A Nitpicker's guide to a minimal-complexity secure GUI. *21st Annual Computer Security Applications Conference (ACSAC)*, 2005; 85–94.
44. Gilad Y, Herzberg A, Trachtenberg A. Securing Smartphones: A Micro-TCB Approach. *ArXiv e-prints* Jan 2014; .
45. Rasmusson L, Corcoran D. Performance overhead of KVM on Linux 3.9 on ARM Cortex-A15. *Proceedings of VtRES: Workshop on Virtualization for Real-Time Embedded Systems*, 2013.
46. Xu Y, Bruns F, Gonzalez E, Traboulsi S, Mott K, Bilgic A. Performance evaluation of para-virtualization on modern mobile phone platform. *Proc. of International Conference on Computer, Electrical, and Systems Science, and Engineering*, 2010; 237–244.
47. Qu J, Li T, Dang F. Performance evaluation and analysis of OpenVPN on Android. *Proc. of Fourth International Conference on Computational and Information Sciences* 2012; **0**:1088–1091, doi:http://doi.ieeecomputersociety.org/10.1109/ICCIS.2012.203.
48. Mayrhofer R, Gellersen H. Shake well before use: Authentication based on accelerometer data. *Proc. Pervasive 2007: 5th International Conference on Pervasive Computing*, *LNCS*, vol. 4480, Springer-Verlag: Berlin, Heidelberg, Wien, 2007; 144–161.