Building Trusted Path on Untrusted Device Drivers for Mobile Devices

Wenhao Li[1], Mingyang Ma[1], Jinchen Han[2], Yubin Xia[1], Binyu Zang[1],
Cheng-Kang Chu[3], Tieyan Li[3]

[1]*Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University, China*
[2]*Software School, Fudan University, China*
[3]*Huawei Technologies Pte Ltd, Singapore* *

Abstract

Mobile devices are frequently used as terminals to interact with many security-critical services such as mobile payment and online banking. However, the large client software stack and the continuous proliferation of malware expose such interaction under various threats, including passive attacks like phishing and active ones like direct code manipulation. This paper proposes TrustUI, a new *trusted path* design for mobile devices that enables secure interaction between end users and services based on ARM's *TrustZone* technology. TrustUI is built with a combination of key techniques including cooperative randomization of the trusted path and secure delegation of network interaction. With such techniques, TrustUI not only requires no trust of the commodity software stack, but also takes a step further by excluding drivers for user-interacting devices like touch screen from its trusted computing base (TCB). Hence, TrustUI has a much smaller TCB, requires no access to device driver code, and may easily adapt to many devices. A prototype of TrustUI has been implemented on a Samsung Exynos 4412 board and evaluation shows that TrustUI provides strong protection of users interaction.

1 Introduction

Mobile devices have been increasingly used to provide users' on-the-go accesses for online shopping, mobile payment and banking transactions. Unfortunately, such convenience comes at a risk, where users' interactions with the Internet services can be easily tampered with or divulged by malware. For example, a victim shopping on Amazon through smartphone can easily be deceived with the amount of money authorized for a transaction (e.g., showing $100 but paid $10,000 instead). Similarly, malware can trick a user entering password on a faked login screen to steal the account information (known as phishing attack).

The key reason with contemporary mobile devices is the lack of *a trusted path* between users and Internet services. On one hand, the architecture of state-of-art mobile system is rather complex, which makes mobile devices an attractive target of attackers. On the other hand, there is usually no effective means to authenticate an user interface to users by ensuring "seeing is believing".

Prior work has attempted to provide a trusted path by hardening different levels in the software and hardware stack, such as operating systems [11, 4, 13, 7], hypervisors [2, 8], and special hardware [9, 10]. While these efforts have made a very good first step, all of these approaches fall shorts in either of the following two aspects, or both. First, some of them need a large TCB (Trusted Computing Base) that includes the entire operating system. However, exploiting root privilege in Android and jailbreak in iOS are not rare in reality, either by rootkit malware, or by the users on purpose. Once an application gains the root privilege, all of the security mechanisms in OS could be easily bypassed. Second, almost all of prior approaches depend on some device-specific user-interacting hardware such as keyboard, display, or touch screen. Hence, prior secure systems usually contain the drivers of these interacting devices, and thus require the drivers to be trusted. However, device drivers are proven

to be vulnerable in mobile, as illustrated in [14] and this further limits the portability of this approach to various kinds of devices, especially in the case where device manufacturers refuse to offer the device driver code.

In this paper, we propose TrustUI, a new design that provides a trusted path between services and end users. TrustUI leverages a widely deployed hardware security feature, named TrustZone, to achieve better security guarantee. By leveraging the split execution mode in TrustZone, TrustUI runs a small and secure kernel in the secure world in parallel with mobile-rich system running in the normal world. To achieve a small TCB, TrustUI takes a step further by excluding the device drivers for input, display and network from the secure world, but instead directly reusing existing drivers from the normal world.

TrustUI is built with several key techniques to provide secure driver reuse, while still provides externally verifiable trust path between users and services. To provide strongly unforgeable user interaction, TrustUI adopts two novel techniques called *input randomization* and *display randomization*, by randomizing input layout and display such that users can easily identify an untampered interaction by comparing them with a tamper-resistant indicator protected by TrustUI. Further, it also adopts cooperative secure channel between the secure world and Internet services to protect the network communication. TrustUI does not rely on the trustworthiness of the entire OS, including device drivers of network, input and display.

We have designed and implemented TrustUI on a Samsung Exynos 4412 development board. To demonstrate the effectiveness of our system, we construct attacks including KeyLogger, ScreenLogger in normal world and the result shows that none of them can bypass TrustUI.

2 Goals and Threat Model

The goal of TrustUI is protecting the interaction between cloud services and end users, while not requiring significant engineering efforts. Specifically, TrustUI aims to achieve the following goals:

First and foremost, the system should provide data confidentiality and integrity for user interaction as well as network communication. It should provide users with verifiable security state to defend against phishing attacks and other tampering. Second, the TCB of our system should be minimal. The OS, including device drivers and network stack, should not be trusted. It should not rely on any specific peripheral devices, e.g., some specific touchscreen or keyboard, in order to be general to different platforms. Third, the system should be deployable to existing mobile devices without significantly modifying existing OSes.

2.1 Threat Model

We trust the hardware of mobile devices, including the TrustZone extension. We cannot defend physical attacks such as reading data directly from the memory. Meanwhile, all of the software running inside the secure world is trusted. On the opposite, the entire OS including device drivers in the normal world is not trusted. We rely on but not trust the services provided by the OS, such as device accessing and network packet sending/receiving. We also do not consider DoS (Deny of Service) attacks issued by the OS, which could be easily figured out by the user. Even if a malicious OS launches a DoS attack, the security properties of the trusted path still remain.

3 Background on TrustZone

Split CPU Mode Execution: TrustZone is a security extension introduced by ARM to achieve strong isolation between secure-sensitive execution environment (secure world) and commodity environment (normal world). The secure world can access all states of normal world but not vice-versa. As shown in Figure 1, there is a higher privilege mode called TrustZone monitor mode (TMM) that is used for switching between the two worlds either by executing the *SMC* instruction or receiving interrupts.
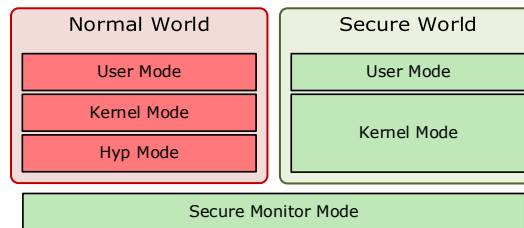

Figure 1: Split CPU Mode with TrustZone Support

Memory and Peripheral Protection: TrustZone supports memory partition between the two worlds. The DRAM could be partitioned into several memory regions by TrustZone Address Space Controller (TZASC), each of which can be configured to be used in either worlds or both. By default secure world applications can access normal world memory but not vice-versa. System peripherals could be configured as secure by TrustZone Protection Controller (TZPC) to ensure these peripherals could only be accessed in secure world. Besides, DMA is also world-awareness and a normal world DMA that transfers data to or from secure memory will be denied.

Interrupt and Exception Isolation: Generic Interrupt Controller (GIC) with TrustZone support can configure an interrupt as secure or non-secure. Secure world,
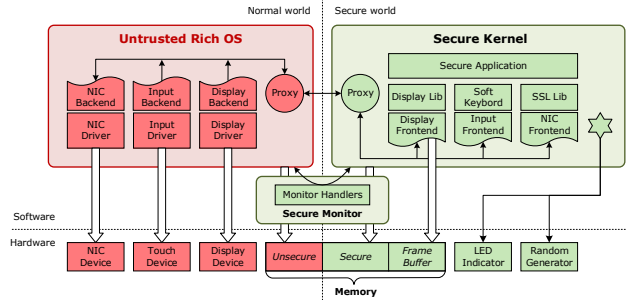
Figure 2: TrustUI overall architecture

Table 2: Security Challenges of TrustUI

Attribute	Attack	Solution
Code Integrity	Code Tampering	Secure Booting
Availability	Denial-of-Service	Detect by user
Display Privacy	Screen-capture	Dedicated FB
Display Integrity	FB Overlay	FB Randomization
Input Privacy	Touch-logger	Input Randomization
Input Privacy	Phishing	LED Indicator
Input Integrity	Fake Input	Random Keyboard

normal world and TMM have their own indepentent exception vector tables and the routing of an interrupt could be configured to one of them. A security memory access violation may cause an external abort and trap to either monitor mode or current CPU state mode exception vector, depending on how the configuration of interrupt behavior is set in secure world.

4 Design

In order to build a trusted path while reusing the device drivers, TrustUI adopts a mechanism that logically splits a device driver into two parts: a backend running in the normal world, and a frontend running in the secure world. The backend part is the unmodified driver and its corresponding wrapper in the normal world, while the frontend part works on top of it and provides safe access to device for secure apps. The two parts communicate through corresponding proxy modules running in both worlds which exchange data through shared memory. The architecture is shown in Figure 2.

A commodity operating system (such as Android) runs in the normal world, while a small *secure kernel* runs in secure world. The secure kernel isolates itself from the normal world by reserving a region of memory for the secure world only. Meanwhile, the secure world can access the entire memory space of normal world if needed. A *monitor handler* that runs in TrustZone monitor mode is responsible for world switch, handling external interrupts and verifying security exceptions. The normal world proxy is a kernel module that runs in the untrusted commodity operating system. It delivers any calls from the *secure kernel* to corresponding backend drivers and transfers the response back. There are three pairs of frontend/backend drivers for input, display and network devices.

TrustUI provides several necessary interfaces between two worlds and Table 1 shows the API list. The input backend driver uses *send_input_data* to deliver the coordinate of the screen to the front driver in secure world.

For display, the frontend driver asks for a framebuffer of the display device from the backend driver, and sets the region of memory as secure-only. Then the secure kernel can operate the framebuffer to display. For network communication, all of the SSL encryption and decryption are done in the secure world, thus the NIC backend only get the encrypted packets to deliver to the NIC.

TrustUI relies on three hardware features. The first one is an LED indicator peripheral with one or more LED lights that can be controlled by the secure world only. The second one is a cryptography component that has a hardware random number generator used to generate unpredictable random seed so as to provide strong entropy since the display and network security of TrustUI highly depends on it. The third one is secure boot process, which uses cryptographic checks to each stage of boot process to ensure the integrity of the loaded code and the root public key is fused in the SoC ROM which is read-only. All of the three hardware features are prevalent on current mobile devices.

4.1 Security Challenges

Giving the overall design above, there are several kinds of potential attacks that may lead TrustUI to an insecure state need to be addressed, as shown in Table 2.

Tampering with secure code: System image usually locates in external storage like flash, attackers from normal world could tamper with the secure code and replace it with an untrusted or even malicious one.

Denial-of-Service attack: Switching from normal world to secure world is invoked proactively via sending requests to *normal world Proxy*. The normal world may refuse to switch to secure world, drop users' input events, or offer a fake framebuffer to the secure kernel.

Screen-capture attack: Attackers may try to capture the display framebuffer to get the confidential display data, thus users' secure sensitive data could be leaked.

Framebuffer overlay attack: A malicious display driver can pass a pointer of framebuffer with low priority to the secure world, and operate on a higher layer framework to overwrite the data user eventually sees. Such attack is also known as "screen hijacking".

Touch-logger attack: Even if the attacker doesn't

Table 1: Interfaces provided by two worlds

World	Command and parameters	Description
Secure World API	*start_secure_app(appid,parameters)* *send_input_data(x,y,z)*	start a secure application in secure world send the touch input data to secure world
Normal World API	*create_shared_buffer(p_addr,size)* *lock_display()* *invoke_ns_func(function_id,parameters,response)* *get_trustui_info()*	create a shared non-secure memory buffer used by both worlds disable modification to display controller and framebuffer in normal world invoke normal world functions get the display information and input interrupt NO from normal world, input interrupt NO is needed so as to keep it unmask in secure world.

know the content of the display, he can still infer user's input by three information: the positions, the times, and the frequency of user's touch operation. Further, attackers may even leverage side channel attacks like motion [3] to infer some sensitive information.

Phishing attack: Invoking secure user interaction is through well-defined interface. Attackers may refuse to switch to secure world but claim that the switch is OK and display a fake secure PIN input screen to cheat user to enter PIN number.

Fake input data attack: An attacker may change the user inputs or fake problematic data to the secure world. For example, when a user touches the screen in position x, the malicious driver may return a wrong position y, thus leading to unexpected result.

The **Denial-of-Service** attack could happen since switching from normal world to secure world is invoked proactively via sending requests to *normal world Proxy* and the normal world may refuse to switch to secure world, drop user's input events, or offer a fake framebuffer to the secure kernel. This kind of attack will not leak any sensitive data and is out of scope of the threat model.
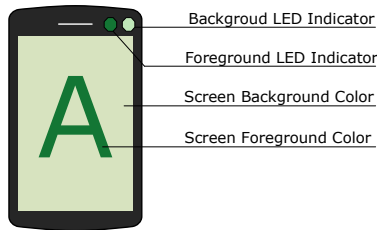
4.2 LED & Display Color Randomization

Figure 3: Use LED to show foreground/background colors

Once the secure kernel gets a framebuffer from the normal world, the display controller and the framebuffer are set as not accessed by normal world.

To prevent framebuffer overlay attack, we leveraged the LED indicator and a new scheme called *color randomization* to bind the content on the screen with the

state of LED indicator. As shown in Figure 3, a secure display contains foreground layer and background layer, both layers render an unpredictable and periodically changed random color. The indicator has one or two LED lights, which can display multiple colors under the control of the secure kernel and will be turned on in secure world only. Their colors are used to indicate the correct foreground and background colors on the screen, respectively. By comparing the display colors with the LED colors, users can confirm that the content on the screen is secure and not overlayed by untrusted normal world driver using hardware display layer.

With TrustUI, users could easily detect Denial-of-Service attack, and would not be cheated because the execution world is identified by the LED Indicator. Screen-capture attack will not work because the framebuffer is set to be secure. Framebuffer overlay attack, meanwhile is more stealthy and the UI scheme color randomization could solve this problem and provide users with a verifiable security state.

4.3 Software Keyboard Randomization

User's inputs are first obtained by the backend driver and then delivered to the frontend driver. As the backend driver is untrusted, this approach may lead to sensitive information leakage or tampering. TrustUI introduces randomization in the generation of software keyboard. To achieve both security and good user experience, we leverage a para-randomized keyboard whose keys are ordered alphabetically as shown in Figure 4 (a). We only choose the position of button 0 randomly, then put all other buttons according to it. Each time the user enters a character, TrustUI regenerates the keyboard by picking up another position for button 0. (Figure 4 (b))

With this keyboard, attackers have no way to know what the user is typing, but he may still learn the length of user's password. Further, TrustUI applies another randomness in soft keyboard: TrustUI will generate some random buttons on the screen within the keyboard area, and ask the user to click the button before continue. As such buttons are standout, it is easier for user to find their positions. However, this makes it possible for attackers to guess which input is injected by analyzing the time

4

interval of two consecutive events. Thus, TrustUI introduces a little delay before showing this button on the screen, the delay time is calculated using user's previous input time interval.
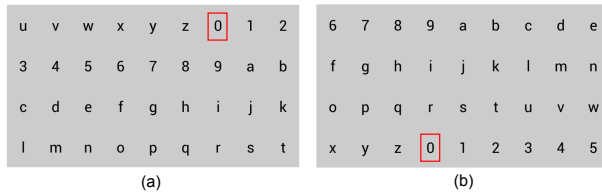

Figure 4: Alphabetic keyboard with random position. The initial look of soft keyboard could be like (a), where the red rectangle indicates the start position of the first key. After user hits one button, the whole keyboard could be re-generated as (b)

In order to defend against fake key injection attack, a secure application could choose to take a *confirm* process by showing some random positioned buttons asking for user's click to confirm the inputs are actually from user.

4.4 Network Delegation

Secure network should be considered so as to achieve a trusted path between application and cloud services. TrustUI adds an SSL library to *secure kernel* and delegates all network stack function calls to *normal world proxy* and call them in untrusted OS. Since all the security related logics (the SSL protocol) is implemented in *Secure Kernel*, no information leakage could occur.

One problem is that the TrustZone itself does not contain secure storage. However, as TrustZone can be used to configure one peripheral as secure world accessible only, it is up to the vendors or TrustZone software developers to decide which peripheral is available for secure storage. Currently TrustUI stores its CA public keys in the secure kernel image. Since public keys do not need to be kept confidential but need to ensure the integrity, with the integrity check enabled in secure boot, the integrity at the boot stage could be guaranteed. After loading the public keys into RAM, the memory region that stores them will be set as secure. In this way, the integrity of public keys could be guaranteed.

5 Preliminary Implementation

The board we use is Samsung Exynos 4412 development board, which has quard-core ARM Cortex-A9 processor with 1.4GHz main frequency and TrustZone support. The reason why we don't use a real phone is that most of TrustZone enabled phones are locked in bootloader or have a secure boot, preventing us from access-

ing the secure world. The development board, on the other hand, is TrustZone unlock and thus we could replace the original bootloader with our customized one. The rich system we run in normal world is Android Ice Cream Sandwich (Android 4.0 version), with Linux kernel version 3.0.2. We implement TrustUI based on T6 [1], a kernel based on ARM ported Xv6 with TrustZone support. In this section, we describe: 1) the boot process, 2)display data protection, and 3)touch input configuration.

5.1 System Boot

Devices with TrustZone support will start in secure world and run a secure world bootloader after running two vender-specific bootloaders. The secure bootloader will load TrustUI image and check the signature of the image using its embedded public key to ensure the integrity of TrustUI before executing the *secure kernel*. Secure kernel will setup the secure environment, such as partition the secure and non-secure memory in TZASC, set the LED Indicator peripheral as secure peripheral in TZPC, configure FIQ as secure interrupt and IRQ as non-secure interrupt in GIC and set the trap behavior to monitor mode when security exception occurs. After configuration, TrustUI will switch to normal world to execute the normal world bootloader and boot Android.

5.2 Protect Display Data

When a user starts a secure interaction, the normal world proxy will switch to secure world together with framebuffer information. The framebuffer memory region and display controller permission will then be set to secure world accesses only by setting TZASC and TZPC, then the LED indicator will be turned on indicating the secure state. When displaying, the display randomization will be employed: the color value of foreground and background layer will be generated by the randomization module and the secure display content is the combination of the two disinct layers, together with the LED Indicator to provide users with security status. Each layer is composed by its own kinds of elements. The background layer elements include bitmap image, pure pixel color and the foreground layer elements include bitmap image, button image and text font. When a processor tries to modify the configuration of display controller or the framebuffer in normal world, an external abort to secure monitor mode will be signaled. Since the trap rate is relatively high sometimes, we modify the normal world untrusted OS by adding a function *lock_display* to lock any modifications to the display controller and framebuffer by instrumenting several lines of code in Android's SurfaceFlinger process to cache the modifications until the

secure world execution ends.

5.3 Configure Touch Input Setting

When a user starts a secure interaction, the normal world proxy will also tell *secure kernel* the interrupt id of input so that *secure kernel* can mask all other IRQs and set the input interrupt to trap to monitor mode by configuring SCR register and GIC, and the *monitor handler* will switch to normal world and get the input data. We use the *getevent* tool in Android to read input data and input backend will transfer the data to secure world to process. To avoid the case that it may never come back to secure world, we set a secure timeout value using a secure timer so that when timeout, the generation of a secure timer interrupt will force world switch back to secure world and *secure kernel* will abort the interaction.

6 Preliminary Evaluation

In this section, we construct and analyze several normal world attacks to evaluate the security of TrustUI. The attacks we construct and analyze include touch-logger and screen-capture attack, screen overlay attack.
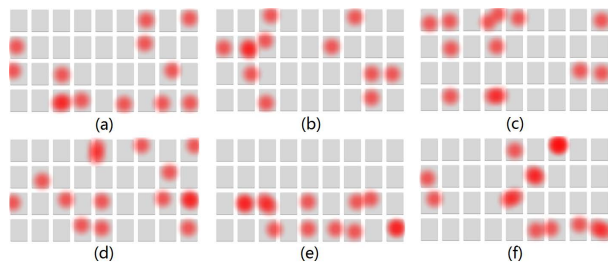
Figure 5: Touch map of different password inputs. Grey rectangles represent keys, and each red circle means one time of click. The opacity indicates the hit count, and multiple hits at one place will cause a deeper red. Figure (a) (b) (c) are got by entering the password '00000000' separately and (d) (e) by '5cfc912f' and '12345678' separately; Figure (f) is generated with 'f6b0736c3b'.

Touch-logger Attack: The input randomization mechanism can ensure the randomness of input coordinates, times, and intervals. As shown in Figure 5, the result of (a)(b)(c) shows that even if the password consists only one kind of character, the input pattern and the length of password are still random. By comparing the difference between (a)(b), (a)(d) and (a)(e), we show that input patterns have no obvious relationship with the content and strongness of the password. For two passwords that have different lengths in (d) and (f), they both need 15 times of click, and it's hard to tell which one has a longer length.
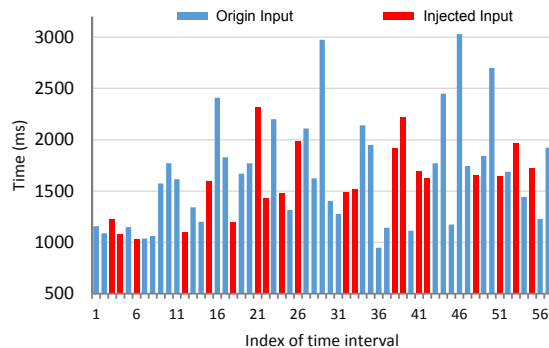
Figure 6: Time intervals between two consective inputs. The columns are arranged as time order, where the red ones are injected inputs and blue ones are normal inputs.

To demonstrate that attackers cannot distinguish the injected inputs from others, we logged the time interval between two clicks with a sentence that contains 37 characters. The actual total number of clicks needed to enter this sentence is 58, counting the injected inputs. The result in Figure 6 shows that neither the longest(which is 3030ms) nor the shortest(which is 948ms) time interval is caused by injected input, and there is no fixed pattern for injected inputs, thus attackers are not able to distinguish these two types of inputs.

Screen-capture Attack: Screen-capture in normal world could be done by reading device file in */dev/graphics/fb0* and some overlay devices like video playback and camera preview. Android provides a tool called *screencap* for easy screen capturing, we use *screencap* with or without display controller and framebuffer locked by *lock_display()*. In both case, *screencap* cannot get the secure display data but the display content before switching to secure world.

Screen Overlay Attack: If an attacker wants to successfully cheat the user with screen overlay attack, he must guess the color of both foreground and background in the secure world correctly. The RGB in the LED light has large color range of 256^3, two colors make the range larger. Though users cannot distinguish two colors with very small difference (in our experience, a difference of 10 could be easily recognized), the possible color range is still large enough. Furthermore, TrustUI will periodically change the foreground and background colors, which makes the guessing harder.

Reduction in TCB: Since TrustUI doesn't need to trust the rich OS or any device drivers, its TCB is larged shrunk. The TCB contains only three frontend drivers, SSL library, the monitor handler, some initialization code, and some other libraries. The entire lines of code is around 10K.

7 Related Work

There are intensive previous researches attempting to provide trusted path. Most of them trust both OS and entire libraries (such as Android framework) and have a large TCB [11, 4, 13, 7], or leverage a hypervisor to provide a trusted path [2, 8, 15], which is not widely available in mobile devices and they include most of the device driver codes in their TCBs. Besides, some systems leverage special hardware to ensure the trusted path and achieve a small TCB [9], but they cannot ensure the security of user interaction. There are several systems that are closed to TrustUI either in their goal or technique. Crossover [5] leverages a hypervisor to provide a UI framework for multiple VMs under mobile virtualization environment. VeriUI [6] runs a Linux in TrustZone secure world to provide an attested login for users, which has a large TCB. TLR [12] provides a framework for running trusted applications on smartphones by splitting an application into secure world and normal world part, but doesn't not provide a trusted path for user interaction. Instead, TLR envisions future hardware feature to achieve the same goal of TrustUI.

8 Summary and Future Work

In this paper, we proposed TrustUI, a system aiming at providing *trusted path* for mobile devices. TrustUI enables secure interaction between end users and services based on *TrustZone*. TrustUI is built with a combination of key techniques including cooperative randomization of the trust path and secure delegation of network interaction. With these techniques, TrustUI not only requires no trust of the commodity software stack, but also takes a step further by excluding drivers for user-interacting devices like touch screen from its TCB. Hence, TrustUI has a much smaller TCB, requires no accesses to device driver code, and can easily adapt to many devices.

In the future, we plan to 1)support multiple secure applications and introduce a tiny sandbox mechanism to isolate them, 2) port several real world secure-sensitive applications running in TrustUI and do a comprehensive security and performance study, 3)support multicore so that TrustUI can run on any cores of a device.

References

[1] T6, an operating system for trustzone based trusted execution environment (tee) in arm-based systems. `http://www.liwenhaosuper.com/projects/t6`.

[2] K. Borders and A. Prakash. Securing network input via a trusted input proxy. In *Proc. USENIX HotSec*, 2007.

[3] L. Cai and H. Chen. Touchlogger: inferring keystrokes on touch screen from smartphone motion. In *Proc. Usenix HotSec*, 2011.

[4] M. Jakobsson and H. Siadati. Spoofkiller: You can teach people how to pay, but not how to pay attention. In *IEEE Workshop on Socio-Technical Aspects in Security and Trust*, 2012.

[5] M. Lange and S. Liebergeld. Crossover: secure and usable user interface for mobile devices with multiple isolated os personalities. In *Proc. ACSAC*, pages 249–257. ACM, 2013.

[6] D. Liu and L. P. Cox. Veriui: Attested login for mobile devices. In *Mobile Computing Systems and Applications, 2007. HotMobile 2014. Eighth IEEE Workshop on*. IEEE, 2014.

[7] D. Liu, E. Cuervo, V. Pistol, R. Scudellari, and L. P. Cox. Screenpass: Secure password entry on touchscreen devices. In *Proc. MobiSys*, 2013.

[8] L. Martignoni, P. Poosankam, M. Zaharia, J. Han, S. McCamant, D. Song, V. Paxson, A. Perrig, S. Shenker, and I. Stoica. Cloud terminal: secure access to sensitive applications from untrusted systems. In *Proc. USENIX ATC*, 2012.

[9] J. M. McCune, B. J. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. Flicker: An execution infrastructure for tcb minimization. In *OS Review*, volume 42, pages 315–328. ACM, 2008.

[10] J. M. McCune, A. Perrig, and M. K. Reiter. Safe passage for passwords and other sensitive data. In *NDSS*, 2009.

[11] Microsoft. How interactive logon works. http://technet.microsoft.com/en-us/library/cc780332 (v=ws.10).aspx, Jan 2009.

[12] N. Santos, H. Raj, S. Saroiu, and A. Wolman. Using arm trustzone to build a trusted language runtime for mobile applications. In *Proc. ASPLOS*. ACM, 2014.

[13] T. Tong and D. Evans. GuarDroid: A Trusted Path for Password Entry. pages 1–10, Apr 2013.

[14] X. Zhou, Y. Lee, N. Zhang, M. Naveed, and X. Wang. The peril of fragmentation: Security hazards in android device driver customizations. In *IEEE Symposium on Security and Privacy*, 2014.

[15] Z. Zhou, V. D. Gligor, J. Newsome, and J. M. McCune. Building verifiable trusted path on commodity x86 computers. In *IEEE Symposium on Security and Privacy*, 2012.