input: 2015 IEEE Conference on Computer Communications (INFOCOM)

2015 IEEE計算機通信會議（INFOCOM）

input: SMOC: A Secure Mobile Cloud Computing Platform

SMOC：安全的移動雲計算平台

input: Zijiang Hao, Yutao Tang, Yifan Zhang. , Ed Novak, Nancy Carter, Qun Li {hebo, yytang, ejnovak, njcarter, liqun}@cs.wm.edu, .zhangy@binghamton.edu College of William and Mary, Williamsburg, VA, USA .State University of New York at Binghamton, Binghamton, NY, USA

紫江浩，湯玉濤，張一凡。 ，Ed Novak，Nancy Carter，Qun Li {hebo，yytang，ejnovak，njcarter，liqun}@cs.wm.edu，.zhangy @ binghamton.edu威廉和瑪麗學院，威廉斯堡，美國弗吉尼亞州紐約州立大學 在Binghamton，Binghamton，NY，USA

input: Abstract—Mobile devices are now ubiquitous in the modern world. In this paper, we propose a novel and practical mobile-cloud platform for smart mobile devices. Our platform allows users to run the entire mobile device operating system and arbitrary applications on a cloud-based virtual machine. It has two design fundamentals. First, applications can freely migrate between the user’s mobile device and a backend cloud server. We design a file system extension to enable this feature, so users can freely choose to run their applications either in the cloud (for high security guarantees), or on their local mobile device (for better user experience). Second, in order to protect user data on the smart mobile device, we leverage hardware virtualization technology, which isolates the data from the local mobile device operating system. We have implemented a prototype of our platform using off-the-shelf hardware, and performed an extensive evaluation of it. We show that our platform is efficient, practical, and secure.

摘要？移動設備在現代世界中無所不在。在本文中，我們提出一個新的和實用的移動雲平台的智能移動設備。我們的平台允許用戶在基於雲的虛擬機上運行整個移動設備操作系統和任意應用程序。它有兩個設計基礎。首先，應用可以在用戶的移動設備和後端雲服務器之間自由地遷移。我們設計了一個文件系統擴展以啟用此功能，因此用戶可以自由選擇在雲中（為了高安全性保證）或在其本地移動設備上運行他們的應用程序（以獲得更好的用戶體驗）。其次，為了保護智能移動設備上的用戶數據，我們利用硬件虛擬化技術，其將數據與本地移動設備操作系統隔離。我們使用現成的硬件實現了我們平台的原型，並對其進行了廣泛的評估。我們展示我們的平台是高效，實用，安全。

input: I. INTRODUCTION Smart mobile devices are gradually becoming the dominant daily computing platform for many people [1], [2]. While many applications today run directly on individual mobile devices, we envision a mobile-cloud computing model emerging whereby individual devices run user interface software with the bulk of computation performed on a virtual machine (VM) running in a commodity cloud hardware environment [3]. In this paper, we aim to build a platform that supports free migration of apps between smart mobile devices and cloud based hardware.

一，引言智能移動設備正逐漸成為許多人的主流日常計算平台[1]，[2]。 雖然目前許多應用直接在單個移動設備上運行，我們設想一種移動雲計算模型，其中個體設備運行用戶接口軟件，在商用雲硬件環境中運行的虛擬機（VM）上執行大量計算[3] 。 在本文中，我們的目標是建立一個平台，支持智能移動設備和基於雲的硬件之間的應用程序的自由遷移。

input: In the mobile-cloud model, user input on the mobile device is transmitted to cloud processes running on the VM. Results from cloud processing are transformed into display content and then transmitted back to the mobile device. The mobile device and the cloud VM share functionality to meet user needs. By moving heavy computational processes from the mobile device to the cloud VM, the mobile-cloud model improves user response time and reduces device energy consumption. It also has security advantages, e.g., sensitive data can be stored in the cloud, safeguarded from a compromised mobile device OS or app, and also protected from exposure to a thief. A user could acquire a new smart mobile device, download the interface software, and resume arbitrary tasks from before the compromise or theft occurred.

在移動雲模型中，移動設備上的用戶輸入被傳送到在VM上運行的雲進程。 來自云處理的結果被轉換成顯示內容，然後被發送回移動設備。 移動設備和雲VM共享功能以滿足用戶需求。 通過將沉重的計算過程從移動設備移動到雲VM，移動雲模型改善了用戶響應時間並減少了設備能量消耗。 它還具有安全優勢，例如，敏感數據可以存儲在雲中，從受危害的移動設備OS或應用保護，並且還防止暴露於小偷。 用戶可以獲得新的智能移動設備，下載接口軟件，並且在發生危害或盜竊之前恢復任意任務。

input: There are already solutions for the mobile-cloud model in recent literature. We believe, however, that our platform has greater potential than what is immediately obvious and can achieve more than what existing solutions can do. There are two concepts underlying our platform that differentiate it. First, we are proposing a resource sharing platform in the sense that an app can freely change its executing location between the mobile

在最近的文獻中已經有針對移動雲模型的解決方案。 然而，我們認為，我們的平台具有比立即顯而易見的更大的潛力，並且可以實現超過現有解決方案可以做到的。 我們的平台有兩個概念來區分它。 首先，我們提出了一個資源共享平台，在一個意義上，應用程序可以自由地改變其在移動之間的執行位置

input: device and the cloud. In contrast, existing solutions only allow apps to run in the cloud. Second, our platform provides security guarantees even when the mobile device operating system has been compromised, which is a feature that existing solutions cannot offer.

設備和雲。 相比之下，現有的解決方案只允許應用程序在雲中運行。 第二，我們的平台即使在移動設備操作系統受損時也提供安全保證，這是現有解決方案不能提供的一個特性。

input: We achieve the first concept by running a VM in the cloud which has an execution environment compatible with that on the smart mobile device. Our platform shares resources between this cloud VM and the mobile device in both directions, i.e., the mobile device shares its files and I/O devices with the cloud VM when the app is running in the cloud, and the cloud VM shares its files with the mobile device when the app is running on the mobile device. The cloud VM does not need to share its (virtual) I/O devices with the mobile device in the latter case, because they are not involved in the app’s execution. Our system currently only supports offline migration, i.e., when an app wants to change its executing location, our platform will cease its execution if it is running, transfer all the related data to the target location, and re-launch the app if necessary. We leave online migration to future work.

我們通過在雲中運行具有與智能移動設備上的執行環境兼容的執行環境的VM來實現第一概念。 我們的平台在這兩個方向共享雲VM和移動設備之間的資源，即移動設備共享其文件和I

input: Considering the first concept, our platform is clearly different from most existing solutions, such as Chrome OS [4]. Like many others, Chrome OS follows a client-server computing model in which the cloud behaves as a server that hosts apps, while the smart mobile device behaves as a client that communicates with the cloud. This model fails to meet a wide range of user needs, because it only allows apps to run in the cloud. Our platform, on the other hand, is a resource sharing platform, in which apps can run on both locations and can freely change their executing location.

考慮到第一個概念，我們的平台顯然不同於大多數現有的解決方案，如Chrome OS [4]。 像許多其他人一樣，Chrome操作系統遵循客戶端 - 服務器計算模式，其中云作為託管應用程序的服務器運行，而智能移動設備作為與云通信的客戶端運行。 這種模式不能滿足廣泛的用戶需求，因為它只允許應用程序在雲中運行。 另一方面，我們的平台是資源共享平台，其中應用可以在兩個位置上運行並且可以自由地改變其執行位置。

input: The second concept is achieved by leveraging the hardware virtualization functionality on the smart mobile device. To be more specific, we suppose that a hypervisor runs on the smart mobile device, which hosts a guest OS. The guest OS could be malicious, and may launch attacks on the apps that it hosts. The apps may also be malicious, compromised, or attacked by the guest OS. In any case, they may leak sensitive information stored on the smart mobile device or input by the user. However, we trust the hypervisor, because the hypervisor is always much smaller than the guest OS, and can be fully verified through formal verification or manual audit. Moreover, the hypervisor is unlikely to install any third-party applications or libraries, and thus gets rid of many potential risks. We also trust the cloud, under the assumption that it is established by a famous company with high concern for their reputation, such as Amazon, Google, Apple, and Microsoft. These companies

第二個概念是通過利用智能移動設備上的硬件虛擬化功能來實現的。 更具體地，我們假設虛擬機管理程序在承載客戶操作系統的智能移動設備上運行。 客戶操作系統可能是惡意的，並可能對其託管的應用程序發起攻擊。 應用程序也可能是惡意，受損或受到來賓操作系統的攻擊。 在任何情況下，它們可能洩漏存儲在智能移動設備上或用戶輸入的敏感信息。 但是，我們相信管理程序，因為管理程序總是比客戶操作系統小得多，並且可以通過正式驗證或手動審核完全驗證。 此外，管理程序不可能安裝任何第三方應用程序或庫，從而擺脫許多潛在的風險。 我們還相信雲，假設它是由一個著名的公司，其高度關注他們的聲譽，如亞馬遜，谷歌，蘋果和微軟。 這些公司

input: 978-1-4799-8381-0/15/$31.00 c2015 IEEE 2668

978-1-4799-8381-0

input: ?2015 IEEE Conference on Computer Communications (INFOCOM)

？2015 IEEE計算機通信會議（INFOCOM）

input: usually have the technical strength to protect their clouds, and are unlikely to intentionally compromise user privacy.

通常具有保護其云的技術力量，並且不可能故意損害用戶隱私。

input: Under these assumptions, our platform offers security guarantees concerning an untrusted guest OS running untrusted apps. Our novel approach is to make the hypervisor responsible for sharing the smart mobile device’s input interfaces with the cloud, and for blocking hardware input events from traveling to the guest OS. By doing this, we can guarantee that the guest OS can learn nothing about the user’s input, while at the same time remain responsible for sharing the smart mobile device’s output interfaces with the cloud. To the best of our knowledge, we are the first to leverage hardware virtualization to support security reinforcement for smart mobile devices.

在這些假設下，我們的平台提供有關運行不受信任的應用程序的不受信任的來賓操作系統的安全保證。 我們的新穎方法是使管理程序負責與雲分享智能移動設備的輸入接口，並阻止硬件輸入事件傳送到客戶操作系統。 通過這樣做，我們可以保證客戶操作系統可以不了解用戶的輸入，同時仍然負責與雲的智能移動設備的輸出接口共享。 據我們所知，我們率先利用硬件虛擬化來支持智能移動設備的安全強化。

input: The following example elaborates on this idea. Consider a smart mobile device, running a compromised guest OS, which executes a malicious background service that stealthily records the user’s input through a software keyboard on the screen and sends it to a remote, malicious user. In this case, if the user runs a banking app directly on the smart mobile device, or through any existing mobile-cloud solution, her account and password are likely to be leaked to the malicious user. If the user runs the banking app on our platform, however, there is no such security concern. The hypervisor will capture the touch events from the software keyboard and forward them directly to the cloud VM, bypassing the guest OS entirely. Meanwhile, the banking app functions properly on the cloud VM, because it receives the user’s input from the hypervisor.

下面的例子詳細說明了這個想法。 考慮運行受攻擊的客戶操作系統的智能移動設備，其執行惡意的後台服務，其通過屏幕上的軟件鍵盤秘密地記錄用戶的輸入，並將其發送給遠程的惡意用戶。 在這種情況下，如果用戶直接在智能移動設備上或通過任何現有的移動雲解決方案運行銀行應用，則她的帳戶和密碼可能被洩漏給惡意用戶。 但是，如果用戶在我們的平台上運行銀行應用程序，則沒有這樣的安全問題。 管理程序將從軟件鍵盤捕獲觸摸事件，並將它們直接轉發到雲VM，完全繞過客戶操作系統。 同時，銀行應用程序在雲VM上正常工作，因為它從管理程序接收用戶的輸入。

input: To summarize, our contributions in this paper are three-fold.

總而言之，我們在本文中的貢獻是三重的。

input: ‧ We propose a secure mobile-cloud computing platform which is designed as a resource sharing platform in the sense that apps running on it can freely change their executing location. This is a more flexible design than those of existing solutions, and it meets a wider range of user needs. ‧ We are the first to leverage hardware virtualization on the smart mobile device to provide security guarantees in the case that the smart mobile device’s operating system is untrusted. ‧ We implement a prototype system following our platform design, and conduct several experiments on this system. The results demonstrate that our platform is efficient and practical. II. RELATED WORK Mobile offloading. An increasing amount of attention has been invested in mobile offloading recently. CloneCloud [5], [6] partitions a mobile application automatically by using static analysis and dynamic profiling, such that part of the application can be offloaded to the cloud to achieve performance and an energy efficiency improvement. It also provides a runtime system to facilitate the offloading. MAUI [7] also approaches the topic of mobile offloading from the perspective of automatic mobile program partition. The main objective of the partition is to dynamically decide which part of the program should be offloaded to achieve maximum energy savings. COMET [8]

？我們提出了一個安全的移動雲計算平台，它被設計為一個資源共享平台，運行在其上的應用程序可以自由地改變其執行位置。這是比現有解決方案更靈活的設計，它滿足更廣泛的用戶需求。 ？我們是第一個利用智能移動設備上的硬件虛擬化，以在智能移動設備的操作系統不受信任的情況下提供安全保證。 ？我們根據我們的平台設計實現一個原型系統，並在這個系統上進行幾個實驗。結果表明我們的平台是高效和實用的。 II。相關工作移動卸載。近來越來越多的人關注移動卸載。 CloneCloud [5]，[6]通過使用靜態分析和動態分析來自動分區移動應用程序，使得應用程序的一部分可以被卸載到雲以實現性能和能效提高。它還提供了一個運行時系統以方便卸載。 MAUI [7]也從自動移動程序分區的角度接近移動卸載的主題。分區的主要目的是動態決定程序的哪個部分應該卸載以實現最大的能量節省。 COMET [8]

input: implements a runtime system on top of the Dalvik Virtual Machine, the virtual machine used by Android, to allow for simultaneous executions of the same multi-threaded application in several machines. The enabling foundation is a distributed shared memory model. In contrast with these existing works, our platform is a resource sharing platform through which the user can run an app on either the cloud or the smart mobile device. Thin client architecture in mobile computing. When a user decides to run a mobile application in the cloud with our system, we essentially turn the local mobile device into a thin client. MobiDesk [9] introduces a mobile virtual desktop hosting infrastructure that transparently decouples the display, operating system, and the network of a user’s computing session from end-user mobile devices, and moves them to hosting providers. SmartVNC [10] is a system to port VNC, which is a remote computing solution, to smartphones while achieving the same level of user experience as on PCs. While work like MobiDesk and SmartVNC try to reduce workloads in mobile devices by turning the devices into thin clients, other works have also studied the energy consumption implication of applying the thin client architecture in mobile computing [11], [12]. Distinct from these works, our platform adopts a spilt client design, combined with hardware virtualization on the smart mobile device, to provide security guarantees even if the smart mobile device’s guest OS is untrusted. Exploiting hardware virtualization technology. Hardware virtualization technologies on the x86 architecture [13], [14] have long been used to develop solutions to protect system security. For example, SecVisor [15] utilizes AMD-V [14] (formally known as AMD Secure Virtual Machine (SVM)) to build a hypervisor with a small code base to ensure the code integrity of guest OS kernels. Specifically, the hypervisor by SecVisor is able to prevent code injection attacks by allowing only user-approved code to run in kernel mode. TrustVisor

在Dalvik虛擬機（Android使用的虛擬機）之上實現一個運行時系統，以允許在多個機器中同時執行相同的多線程應用程序。啟用基礎是一種分佈式共享內存模型。與這些現有作品相反，我們的平台是一個資源共享平台，通過它，用戶可以在雲或智能移動設備上運行應用程序。移動計算中的瘦客戶端架構。當用戶決定使用我們的系統在雲中運行移動應用程序時，我們基本上將本地移動設備變成瘦客戶端。 MobiDesk [9]介紹了移動虛擬桌面託管基礎設施，透明地將顯示器，操作系統和用戶計算會話的網絡與終端用戶移動設備分離，並將其移動到託管提供商。 SmartVNC [10]是一個將VNC（一種遠程計算解決方案）連接到智能手機，同時實現與PC相同的用戶體驗水平的系統。雖然像MobiDesk和SmartVNC這樣的工作嘗試通過將設備轉換為瘦客戶端來減少移動設備中的工作負載，但其他工作也研究了在移動計算中應用瘦客戶端架構的能耗影響[11,12]。與這些作品不同，我們的平台採用溢出的客戶端設計，結合智能移動設備上的硬件虛擬化，即使智能移動設備的客戶操作系統不受信任，也能提供安全保證。利用硬件虛擬化技術。 x86架構上的硬件虛擬化技術[13]，[14]長期以來一直用於開發解決方案以保護系統安全。例如，SecVisor [15]利用AMD-V [14]（正式稱為AMD安全虛擬機（SVM））構建具有小代碼庫的管理程序，以確保客戶機操作系統內核的代碼完整性。具體來說，SecVisor的管理程序能夠通過僅允許用戶批准的代碼在內核模式下運行來防止代碼注入攻擊。 TrustVisor

input: [16] also uses AMD-V to build a small hypervisor to ensure the guest OS’s data integrity and secrecy, in addition to code integrity. Lares [17] exploits Intel VT-x [13] to achieve active security monitoring inside a virtual machine environment. In our platform, we choose to allow the mobile device OS and apps to run on VMs in the cloud. Part of the reason for this choice is to utilize these existing solutions to ensure we have a secure execution environment in the cloud. On the local device side, we exploit the newly developed ARM hardware virtualization technology [18] to protect the user’s input and data collected by various on-board sensors, which can reveal sensitive information about the user. Although there are some efforts to use this technology to build general purpose hyper-visors [19], [20], to the best of our knowledge, we are the first to utilize it to build a system specifically for security/privacy. III. PLATFORM DESIGN Our platform spans across the user’s local mobile device and a remote cloud server. We believe that such a mobile-cloud computing platform is more capable than what people usually

[16]還使用AMD-V構建一個小型管理程序，以確保客戶操作系統的數據完整性和保密性，以及代碼完整性。 Lares [17]利用英特爾VT-x [13]在虛擬機環境中實現主動安全監控。 在我們的平台中，我們選擇允許移動設備操作系統和應用程序在雲中的虛擬機上運行。 這種選擇的部分原因是利用這些現有的解決方案，以確保我們在雲中有一個安全的執行環境。 在本地設備端，我們利用新開發的ARM硬件虛擬化技術[18]來保護用戶的輸入和各種板載傳感器收集的數據，從而揭示用戶的敏感信息。 雖然有一些努力使用這種技術來構建通用的超級面罩[19]，[20]，據我們所知，我們是第一個利用它來構建專門用於安全的系統

input: 2669

2669

input: ?2015 IEEE Conference on Computer Communications (INFOCOM)

？2015 IEEE計算機通信會議（INFOCOM）

input: think and can achieve more than what existing solutions can do. In this section, we describe our platform design in detail.

認為和可以實現比現有解決方案可以做到的更多。 在本節中，我們詳細描述我們的平台設計。

input: A. Design goals The main design goal of our platform is to share the necessary resources between the smart mobile device and the cloud, such that an app can run either on the smart mobile device or in the cloud, arbitrarily. This design goal contains two aspects. First, the platform must be capable of executing an app in the cloud without any modifications to the app itself. This is possible only if the platform shares the smart mobile device’s I/O interfaces with the cloud, such that an app running in the cloud can receive the user’s input (e.g., from touch screen, sensors, keyboard, etc.) and render the output (e.g., display, sound, etc.) on the mobile device. Second, an app installed in the cloud should be able to be downloaded to the smart mobile device and work properly, as it does in the cloud. This can be achieved if the app can access the same resources regardless of its location. For example, files accessed by the app should be synchronized across both locations.

A.設計目標我們的平台的主要設計目標是在智能移動設備和雲之間共享必要的資源，使得應用可以在智能移動設備上或云中任意運行。 這個設計目標包含兩個方面。 首先，平台必須能夠在雲中執行應用程序，而不對應用本身進行任何修改。 只有當平台共享智能移動設備時，這是可能的

input: Running apps in the cloud is necessary for several reasons. First, the user may not trust the app (e.g., because the app has access to sensitive data) but may still want to run it. In this case, she may utilize our platform and run the app on a VM in the cloud. Even if the app compromises this VM, it cannot affect other apps and data on the user’s local mobile device, and the user may simply delete the VM afterwards. In this way, even a malicious app is completely quarantined. Moreover, commercial clouds often provide powerful anti-virus services, and a malicious app is more likely to be caught if these services are implemented in the cloud used in our platform. Second, smart mobile devices are always resource constrained [21]–[23].To solve this problem, our platform allows the user to offload apps from the smart mobile device to the cloud, which typically has abundant resources. Third, some organization or developer may want to publish an app without disclosing any proprietary secrets about it (e.g., the binary of the app might be reverse engineered to compromise some critical algorithms). In this case, the organization/developer may publish the app using our platform and only allow the app to run in the cloud. In this way, no malicious user can recover a complete binary file for further analysis.

在雲中運行應用程序是必要的，有幾個原因。首先，用戶可能不會信任該應用（例如，因為應用程序有權訪問敏感數據），但仍可能希望運行它。在這種情況下，她可以利用我們的平台並在雲中的虛擬機上運行應用程序。即使應用程序損害此VM，也不會影響用戶本地移動設備上的其他應用程序和數據，用戶可以隨後簡單地刪除VM。這樣，即使惡意應用程式也完全隔離。此外，商業云通常提供強大的反病毒服務，如果這些服務在我們的平台中使用的雲中實施，惡意應用程序更有可能被捕獲。第二，智能移動設備總是資源有限的[21]？[23]。為了解決這個問題，我們的平台允許用戶將應用從智能移動設備卸載到雲，這通常有豐富的資源。第三，一些組織或開發人員可能想要發布應用程序，而不披露其任何專有秘密（例如，應用程序的二進制可能被反向工程以損害一些關鍵算法）。在這種情況下，組織

input: After being deployed, our platform will have some apps installed in the cloud by default. However, the user may want to download an app from the cloud to the smart mobile device for some reason, for example, the latency between the smart mobile device and the cloud becomes unacceptable. This can be achieved transparently in our platform, because all resources, including files, are shared between the smart mobile device and the cloud.

部署後，我們的平台默認情況下會在雲中安裝一些應用程序。 然而，由於某些原因，例如，智能移動設備和雲之間的等待時間變得不可接受，用戶可能想要將應用從雲下載到智能移動設備。 這可以在我們的平台中透明地實現，因為所有資源，包括文件，在智能移動設備和雲之間共享。

input: The cloud exploited in our platform is considered a commercial cloud. However, it could also be a private cloud established by, and serving, only a single user. Consider the scenario where a user does not trust an app, but still wants to run it. She may build a private cloud which does not connect to the Internet, and deploy our platform on this cloud in conjunction with her

在我們的平台中利用的雲被認為是商業雲。 然而，它也可以是由單個用戶建立和服務的私有云。 考慮用戶不信任應用程序但仍希望運行應用程序的情況。 她可以構建不連接到互聯網的私有云，並且與她一起在我們的雲上部署我們的平台

input: smart mobile device through a local area network. She can then run the app in the private cloud to prevent her mobile device from being attacked by the app.

智能移動設備通過局域網。 她可以在私有云中運行應用程序，以防止她的移動設備受到應用程序的攻擊。

input: Another design goal of our platform is to provide a secure environment for the user to run apps. As we mentioned previously, the VM in the cloud is isolated from the smart mobile device, and thus a malicious app running on the VM cannot affect any other apps running on the smart mobile device. Moreover, commercial clouds often provide powerful anti-virus services, which can be utilized on our platform to defend against malicious apps. These are two cases in which our platform can provide strong security guarantees. We have yet to consider another important scenario; suppose a user wants to run a banking app on the smart mobile device, but the smart mobile device’s operating system has been compromised and a malicious background service stealthily records the user’s input and sends it to a malicious user. In this scenario, running the banking app on the smart mobile device is dangerous because the malicious user may learn the user’s banking account and password via her input. Our platform provides a secure environment that defends against this attack by leveraging hardware virtualization on the smart mobile device.

我們平台的另一個設計目標是為用戶提供一個安全的環境來運行應用程序。如前所述，雲中的VM與智能移動設備隔離，因此在VM上運行的惡意應用程序不會影響智能移動設備上運行的任何其他應用程序。此外，商業云通常提供強大的反病毒服務，可以在我們的平台上利用來防禦惡意應用程序。這兩種情況下，我們的平台可以提供強大的安全保證。我們還沒有考慮另一個重要的情況;假設用戶想要在智能移動設備上運行銀行應用，但是智能移動設備的操作系統已經被盜用並且惡意的後台服務偷偷地記錄用戶的輸入並將其發送給惡意用戶。在這種情況下，在智能移動設備上運行銀行應用是危險的，因為惡意用戶可以通過她的輸入來學習用戶的銀行賬戶和密碼。我們的平台提供了一個安全的環境，通過利用智能移動設備上的硬件虛擬化來抵禦這種攻擊。

input: B. Design details Hardware Host OS Hypervisor Guest OS APP VM Display, Sound, etc. Touch Screen, Sensors, Keyboard, etc. Cloud Input Proxy Output Renderer Input Injector Output Proxy Smart Mobile Device

B.設計細節硬件主機操作系統管理程序來賓操作系統APP VM顯示，聲音等觸摸屏，傳感器，鍵盤等雲輸入代理輸出渲染器輸入註射器輸出代理智能移動設備

input: Fig. 1: An app is running in the cloud in our platform.

圖。 1：應用程序在我們的平台上的雲中運行。

input: Fig. 1 demonstrates our design when an app is running in the cloud on our platform. The app is running in the cloud, hosted by a VM which has an execution environment compatible with that of the smart mobile device. This allows the app to run without requiring any changes. Hardware virtualization is enabled on the smart mobile device. A hypervisor running on the smart mobile device hosts one or more guest OSes in which the app can be executed.

圖。 1演示了當我們的平台上的應用程序在雲中運行時我們的設計。 該應用程序在雲中運行，由具有與智能移動設備的執行環境兼容的執行環境的VM託管。 這允許應用程序運行，而不需要任何更改。 在智能移動設備上啟用硬件虛擬化。 在智能移動設備上運行的管理程序託管可以在其中執行應用程序的一個或多個客戶操作系統。

input: As described in our main design goal, the smart mobile device needs to share I/O interfaces with the VM such that the app can work properly in the cloud. To achieve this, two client programs are provided on the smart mobile device. One of them, depicted as “Input Proxy” in Fig. 1, is responsible for capturing the user’s input through the smart mobile device’s input interfaces (e.g., touch screen, sensors, keyboard, etc.) and sending it to the VM. The VM will then emulate the user’s input such that the app can receive it and work properly. On the other hand, the app generates output (e.g., display, sound,

如我們的主要設計目標所述，智能移動設備需要共享I

input: 2670

2670

input: ?2015 IEEE Conference on Computer Communications (INFOCOM)

？2015 IEEE計算機通信會議（INFOCOM）

input: etc.) when it is running in the cloud and sends the output to the VM’s output interfaces. The VM will then transfer the app’s output to the smart mobile device. The other program, depicted as “Output Renderer” in Fig. 1, is responsible for receiving this output and rendering it on the smart mobile device so the user can perceive it. By communicating in this way, the smart mobile device shares its I/O interfaces with the cloud VM such that the app can be executed in the cloud instead of on the smart mobile device.

等等），當它在雲中運行並將輸出發送到VM的輸出接口。 VM然後將應用的輸出傳送到智能移動設備。 其他程序，描述為？Output Renderer？ 。 1，負責接收該輸出並將其呈現在智能移動設備上，使得用戶可以感知它。 通過以這種方式通信，智能移動設備共享其I

input: Hardware Host OS Hypervisor Guest OS APP VM Cloud Distributed File System Smart Mobile Device

硬件主機操作系統管理程序來賓操作系統APP VM雲分佈式文件系統智能移動設備

input: Fig. 2: An app installed in the cloud is downloaded to, and executed on, the smart mobile device in our platform.

圖。 2：安裝在雲中的應用程序下載到我們平台中的智能移動設備並在其上執行。

input: Fig. 2 demonstrates our design in the case that an app installed in the cloud is downloaded to, and executed on, the smart mobile device. Our platform allows the user to either launch the app in the cloud or download it to the smart mobile device and execute it. When the user downloads the app to the smart mobile device, a distributed file system across the VM that hosts the app in the cloud and the guest OS on the smart mobile device will be automatically enabled. This distributed file system is needed in order for the VM to share resources with the smart mobile device, allowing the app to work properly without any modifications. Recall that the smart mobile device needs to share its I/O interfaces with the VM when the app is running in the cloud. As the app is now running on the smart mobile device, it can directly access the smart mobile device’s I/O interfaces, so no I/O interface sharing is needed. However, the app, previously installed in the cloud, may need to access some resources (such as files) on the VM to work properly. For this reason, we design the distributed file system in our platform.

圖。 圖2示出了在雲中安裝的應用下載到智能移動設備並在智能移動設備上執行的情況下的我們的設計。 我們的平台允許用戶在雲中啟動應用程序或將其下載到智能移動設備並執行它。 當用戶將應用下載到智能移動設備時，將跨越在雲中託管應用的VM和智能移動設備上的客戶OS的分佈式文件系統將被自動啟用。 需要這種分佈式文件系統，以便VM能夠與智能移動設備共享資源，從而允許應用在沒有任何修改的情況下正常工作。 回想一下，智能移動設備需要共享它的I

input: There are two ways to download an app from the cloud to the smart mobile device. The first way is offline downloading, in which the user downloads the app’s binary file when the app is not running, and executes it on the smart mobile device thereafter. The second way is online downloading, in which the user migrates the app’s process while the app is running, and continues itsexecution on the smart mobiledevice.For simplicity, our platform currently only supports offline downloading. However, conceptually, online downloading could also be achieved. We leave this to future work. After being downloaded to the smart mobile device, the app may need to access files on the VM that previously hosted it. Our distributed file system, spanning across the smart mobile device and the VM, allows the app to

有兩種方法可以將應用程序從雲下載到智能移動設備。 第一種方式是離線下載，其中用戶在應用不運行時下載應用的二進製文件，並且之後在智能移動設備上執行它。 第二種方式是在線下載，其中用戶在應用程序運行時遷移應用程序的進程，並繼續在智能移動設備上執行。為了簡單起見，我們的平台目前僅支持離線下載。 然而，在概念上，也可以實現在線下載。 我們把這留給未來的工作。 在下載到智能移動設備後，應用程序可能需要訪問以前託管它的VM上的文件。 我們的分佈式文件系統，跨越智能移動設備和VM，允許應用程序

input: access such files after it has been downloaded. Therefore, the app can work properly on the smart mobile device without any modifications.

在下載後訪問這些文件。 因此，該應用程序可以在智能移動設備上正常工作，無需任何修改。

input: Combining these two design elements, illustrated by Fig. 1 and Fig. 2, our platform achieves its main goal. The second design goal can also be achieved if the hypervisor in Fig. 1 is trusted. Thisisa reasonable assumption, because thehypervisor is always much smaller than the guest OS, and can be fully verified through formal verification or manual audit. Moreover, the hypervisor is unlikely to install any third-party applications or libraries, and thus gets rid of many potential risks.

組合這兩個設計元件，如圖1所示。 如圖1和圖2所示。 2，我們的平台實現了其主要目標。 第二設計目標也可以實現，如果圖1中的管理程序 1是受信任的。 Thisisa的合理假設，因為hypervisor總是比客戶操作系統小得多，並且可以通過正式驗證或手動審核完全驗證。 此外，管理程序不可能安裝任何第三方應用程序或庫，從而擺脫許多潛在的風險。

input: As we mentioned previously, the guest OS running on the smart mobile device is untrusted in our platform. It might be malicious and runs a stealthy key logger in the background. In this case, the user’s private information, such as the banking account and password, could be compromised. To solve this problem, our platform leverages hardware virtualization on the smart mobile device and adopts a split design for the client program to share the smart mobile device’s I/O interfaces. Fig. 1 demonstrates how our platform provides strong security guarantees in the case that the guest OS is untrusted. The guest OS runs on the trusted hypervisor, rather than directly on the smart mobile device’s hardware. The app runs on a VM in the trusted cloud. The input proxy, residing in the hypervisor, traps the user’s input such that it cannot be received by the guest OS, and transfers it to the VM. This is feasible because the hypervisior is the first layer on the smart mobile device at which hardware events (such as touch points) arrive, so the input proxy can process these events before passing them to the guest OS, or hiding them from the guest OS. Therefore, the app can work properly in the cloud, but the guest OS cannot learn anything about the user’s input. On the other hand, the guest OS is the location where the user “logically” launches the app, so it should also be the location where the app’s output should be rendered. For this reason, the output renderer is placed in the guest OS, responsible for rendering the app’s output.

正如我們之前提到的，智能移動設備上運行的客戶操作系統在我們的平台上是不可信的。 它可能是惡意的，並在後台運行隱身的鍵盤記錄器。 在這種情況下，用戶的私人信息（例如銀行帳戶和密碼）可能被洩露。 為了解決這個問題，我們的平台利用智能移動設備上的硬件虛擬化，並採用分離設計為客戶端程序共享智能移動設備

input: Our platform provides security guarantees even if the guest OS is untrusted, because the VM in the cloud takes the place of the untrusted guest OS. The app runs on the VM instead of the guest OS, and the user’s input is sent to the VM but hidden from the guest OS. However, this implies that our platform only works if the cloud is trusted. We do trust the cloud in our platform, as we mentioned previously, by assuming that the cloud is provided by a famous company with high concern for their reputation, such as Amazon, Google, Apple, and Microsoft. These companies are unlikely to intentionally compromise their users’ privacy. Moreover, their clouds often provide powerful anti-virus services, and thus a compromised VM or app running in the cloud will likely be caught by these anti-virus services.

即使客戶操作系統不受信任，我們的平台也能提供安全保證，因為雲中的VM取代了不受信任的客戶操作系統。 該應用程序在VM而不是客戶機操作系統上運行，並且用戶的輸入發送到VM，但是對客戶操作系統隱藏。 然而，這意味著我們的平台只有在雲是可信的時候才能工作。 我們在我們的平台中信任雲，正如我們之前提到的，假設雲是由一個著名的公司，如亞馬遜，谷歌，蘋果和微軟高度關注他們的聲譽。 這些公司不太可能故意損害他們的用戶？ 隱私。 此外，他們的云通常提供強大的反病毒服務，因此在雲中運行的受損VM或應用可能被這些反病毒服務捕獲。

input: It is also arguable that the user may delete the guest OS if she suspects the guest OS has been compromised, and installs a new one that is safe to execute apps. This solution may also solve the problem, but has some drawbacks. The main drawback results from the fact that a smart mobile device is usually resource

還可以論證的是，如果用戶懷疑客戶操作系統已經被盜用，則用戶可以刪除客戶操作系統，並且安裝一個新的可以安全執行應用的操作系統。 這個解決方案也可以解決問題，但是有一些缺點。 主要缺點源於智能移動設備通常是資源的事實

input: 2671

2671

input: ?2015 IEEE Conference on Computer Communications (INFOCOM)

？2015 IEEE計算機通信會議（INFOCOM）

input: constrained and thus can support only a limited number of guest OSes. In this case, a guest OS on the smart mobile device may have many apps installed and most of them may have no security issues, even if they are executed in a compromised guest OS. Therefore, the user may be unwilling to delete this guest OS. Meanwhile, the limited resources on the smart mobile device prohibit the user from installing a new guest OS.

因此可以僅支持有限數量的客戶操作系統。 在這種情況下，智能移動設備上的客戶OS可以安裝許多應用，並且其中大多數可以不具有安全問題，即使它們在受損的客戶OS中執行。 因此，用戶可能不願意刪除該客戶OS。 同時，智能移動設備上的有限資源禁止用戶安裝新的客戶OS。

input: The second design goal can only be archived by the design illustrated in Fig. 1, which implies that the user needs to execute the app in the cloud to enjoy strong security guarantees. When the user downloads the app to the smart mobile device for some reason, e.g., the latency between the smart mobile device and the cloud becomes unacceptable, she implies that she wants to trade strong security guarantees for higher usability. Therefore, it is natural that our platform does not make the same security guarantees in this case.

第二個設計目標只能通過圖1所示的設計歸檔。 1，這意味著用戶需要在雲中執行應用程序以享受強大的安全保證。 當用戶由於某種原因將應用下載到智能移動設備時，例如，智能移動設備和雲之間的等待時間變得不可接受，她暗示她想要交換強的安全保證以實現更高的可用性。 因此，在這種情況下，我們的平台自然不會提供相同的安全保證。

input: IV. SYSTEM IMPLEMENTATION To prove that our platform is practical and can work well, we implement a prototype system, following the design outlined in the previous section. In this section, we elaborate on the implementation details of this prototype system.

IV。 系統實現為了證明我們的平台是實用的並且可以工作，我們實現一個原型系統，按照上一節中概述的設計。 在本節中，我們詳細介紹了這個原型系統的實現細節。

input: A. Setup The smart mobile device used in the prototype is a Samsung Chromebook. We choose this mobile device because it is one of a few that support hardware virtualization, which is required by our platform. The host OS installed on this Chromebook is Ubuntu Linux and the hypervisor running on this host OS is KVM plus QEMU. Both KVM and QEMU are customized for the Chromebook, as provided by [24], the authors of which also provide a bootloader to enable the hardware virtualization features of the Chromebook. QEMU emulates a Cortex-A15 VExpress hardware abstraction, and Android is installed on this hardware abstraction as the guest OS.

A.設置原型中使用的智能移動設備是三星Chromebook。 我們選擇這種移動設備是因為它是支持硬件虛擬化的少數幾個，這是我們的平台所需要的。 此Chromebook上安裝的主機操作系統是Ubuntu Linux，在此主機操作系統上運行的虛擬機管理程序是KVM plus QEMU。 KVM和QEMU都是為Chromebook定制的，由[24]提供，其作者還提供啟動加載程序以啟用Chromebook的硬件虛擬化功能。 QEMU模擬Cortex-A15 VExpress硬件抽象，Android作為客戶操作系統安裝在這個硬件抽像上。

input: The cloud in the prototype system is established on a Lenovo laptop. The host OS on the cloud is Windows, and VMware Workstation is installed on the host OS as the hypervisor. A VM is created on the VMware Workstation with Android-x86

原型系統中的雲是在Lenovo筆記本電腦上建立的。 雲上的主機操作系統是Windows，VMware Workstation作為虛擬機管理程序安裝在主機操作系統上。 在具有Android-x86的VMware Workstation上創建虛擬機

input: [25] installed as the guest OS. B. Sharing I/O interfaces Fig. 3 demonstrates how the components cooperate in the prototype system when an app is executed in the cloud. Darker blocks indicate the components where our implementation has been involved. As depicted in Fig. 3, four components are implemented to share the I/O interfaces across the smart mobile device and the cloud. They are input proxy, input injector, output proxy, and output renderer.

[25]安裝為客戶操作系統。 B.分享我

input: The input proxy works with the input injector to share the input interfaces across the smart mobile device and the cloud. As described in Section III-B, the input proxy resides in the hypervisor on the smart mobile device. We integrate the input proxy into QEMU in our implementation, because QEMU is part of the hypervisor. More specifically, we implement the input proxy’s functionality in QEMU, such that QEMU can

輸入代理與輸入註入器一起工作以共享智能移動設備和雲上的輸入接口。 如第III-B節所述，輸入代理駐留在智能移動設備上的管理程序中。 我們在我們的實現中將輸入代理集成到QEMU中，因為QEMU是虛擬機管理程序的一部分。 更具體地說，我們在QEMU中實現輸入代理的功能，使QEMU可以

input: Hardware Linux KVM Android APP Android-x86 Samsung Chromebook Exynos 5250 Vmware Workstation Windows Hardware Lenovo IdeaPad Y480 Input Injector Output Proxy Input Output Output Renderer QEMU (integrated with Input Proxy) Fig. 3: An app is executed in the cloud in the prototype system. Darker blocks indicate the components where our implementation has been involved.

硬件Linux KVM Android APP Android-x86三星Chromebook Exynos 5250 Vmware工作站Windows硬件Lenovo IdeaPad Y480輸入註入器輸出代理輸入輸出輸出渲染器QEMU（與輸入代理集成） 3：應用程序在原型系統中的雲中執行。 更深的塊表示我們的實現涉及的組件。

input: capture the user’s input from the keyboard (i.e., the hardware keyboard) as well as an attached accelerometer (SEN-10537 Serial Accelerometer Dongle), and transfer it to the cloud. There is not much difference between the keyboard and the accelerometer from QEMU’s point of view. Therefore, we only focus on the keyboard part in the following discussion.

捕獲來自鍵盤（即，硬件鍵盤）以及附加的加速度計（SEN-10537串行加速計加密狗）的用戶輸入，並將其傳送到雲。 從QEMU的角度來看，鍵盤和加速度計之間沒有太大的區別。 因此，在下面的討論中我們只關注鍵盤部分。

input: As the input proxy is integrated in QEMU, we need to define a way by which QEMU can enter and exit the input proxy mode to emulate the input proxy’s launch and termination. In our implementation, QEMU will enter the input proxy mode when it detects that the user has repeatedly pressed the “Caps Lock” key six times. After entering the input proxy mode, QEMU will continue monitoring the keyboard and exit this mode if it receives an ESC keystroke. When it is in the input proxy mode, QEMU maintains a connection to the cloud. Upon receiving a key event from the host OS, QEMU will encrypt the key code and send it to the cloud. It will not generate the virtual keyboard interrupt for the guest OS. Therefore, the guest OS will not know this key event and thus cannot learn the user’s keyboard input, as described in Section III.

由於輸入代理集成在QEMU中，我們需要定義QEMU可以進入和退出輸入代理模式的方式，以模擬輸入代理的啟動和終止。 在我們的實現中，QEMU將在檢測到用戶反复按下？Caps Lock時進入輸入代理模式？ 鍵六次。 進入輸入代理模式後，QEMU將繼續監控鍵盤，並退出此模式，如果它收到ESC鍵擊。 當它處於輸入代理模式時，QEMU維護與雲的連接。 在從主機OS接收到鍵事件時，QEMU將加密鍵代碼並將其發送到雲。 它不會為客戶操作系統生成虛擬鍵盤中斷。 因此，客戶操作系統不會知道這個鍵事件，因此不能學習用戶的鍵盤輸入，如第三節所述。

input: We have considered several places to implement the input proxy functionality. The first place we have considered is the host OS’skeyboard driver. However, we have quickly found that this is not a good choice, because it affects all the programs residing in the host OS alongside the hypervisor (i.e., any program running on the host OS will not receive any input events from the keyboard). The second place is the guest OS’s keyboard driver. We have also decided against this choice, because it violates our platform design by making the guest OS capable of learning the user’s input when the app is running in the cloud. According to our design in Section III, the most natural place to implement this functionality is in the hypervisor. As QEMU is responsible for providing the hardware abstraction to the guest OS, we have decided to implement this functionality in QEMU.

我們已經考慮了幾個地方來實現輸入代理功能。 我們考慮的第一個地方是主機OS？skeyboard驅動程序。 然而，我們很快發現這不是一個好的選擇，因為它影響駐留在主機OS中的所有程序以及管理程序（即，在主機OS上運行的任何程序將不會從鍵盤接收任何輸入事件）。 第二個是客戶操作系統的鍵盤驅動程序。 我們還決定反對這種選擇，因為它違反了我們的平台設計，通過使客戶操作系統能夠在應用程序在雲中運行時學習用戶的輸入。 根據我們在第三部分中的設計，實現此功能的最自然的地方是在管理程序中。 由於QEMU負責為客戶操作系統提供硬件抽象，我們決定在QEMU中實現此功能。

input: The input injector in the cloud, implemented as an Android app on the Android-x86 VM, is responsible for maintaining the connection to QEMU. It also listens on this connection for encrypted key codes sent by QEMU. When an encrypted key code is received, the input injector will decrypt it, and inject

雲中的輸入註入器實現為Android-x86 VM上的Android應用程序，負責維護與QEMU的連接。 它還監聽此連接以獲取QEMU發送的加密密鑰代碼。 當接收到加密的密鑰代碼時，輸入註入器將對其解密，並註入

input: 2672

2672

input: ?2015 IEEE Conference on Computer Communications (INFOCOM)

？2015 IEEE計算機通信會議（INFOCOM）

input: the key code into the VM through the Linux sendevent utility. Then the app running on the VM will be notified of the key event and receive the key code.

關鍵代碼通過Linux的sendevent實用程序進入VM。 然後，將在VM上運行的應用程序通知鍵事件並接收鍵代碼。

input: The output proxy works with the output renderer to share the output interfaces across the smart mobile device and the cloud. We only consider sharing the screen frames in our prototype system, as this is enough for the user to track the app’s output on the mobile device in most cases. Therefore, the output proxy needs to take a screenshot of the Android-x86 VM periodically and transfer it to the output renderer, which will then render it on the smart mobile device’s screen.

輸出代理與輸出渲染器一起工作，以在智能移動設備和雲之間共享輸出接口。 我們只考慮在我們的原型系統中共享屏幕幀，因為在大多數情況下，這足以讓用戶在移動設備上跟踪應用程序的輸出。 因此，輸出代理需要周期性地截取Android-x86虛擬機的屏幕截圖，並將其傳輸到輸出渲染器，然後將其顯示在智能移動設備的屏幕上。

input: The output proxy is implemented as another Android app alongside the input injector on the Android-x86 VM. It is responsible for maintaining a connection to the output render, and for sending the VM’s screenshots periodically, as fast as possible, through this connection. It captures the VM’s screenshot by invoking the screencap program provided by Android-x86. A captured screenshot is then stored as a png image file on the local disk. Finally, the output proxy sends the file data to the output renderer, which will further display the screenshot on the smart mobile device’s screen.

輸出代理作為另一個Android應用程序與Android-x86 VM上的輸入註入器一起實現。 它負責維護與輸出渲染的連接，並通過此連接定期盡可能快地發送VM的屏幕截圖。 它通過調用Android-x86提供的screencap程序來捕獲VM的屏幕截圖。 捕獲的屏幕截圖隨後作為png圖像文件存儲在本地磁盤上。 最後，輸出代理將文件數據發送到輸出渲染器，這將進一步在智能移動設備的屏幕上顯示截圖。

input: The output renderer is implemented as an Android app running in the Android guest OS on the smart mobile device. It connects to the output proxy when the app is launched in the cloud. After the connection has been established, the output renderer listens to this connection for the VM’s screenshots sent by the output proxy. When a screenshot is received, the output renderer stores it as a local png image file and sets this file as the source of its SurfaceView component. The SurfaceView component then displays the image file on the screen. This process is done periodically, as fast as possible, such that the user will perceive the stream of the app’s screen output as if the app were running locally.

輸出渲染器實現為在智能移動設備上的Android客戶操作系統中運行的Android應用程序。 當應用程序在雲中啟動時，它連接到輸出代理。 在建立連接後，輸出渲染器將偵聽此連接以獲取由輸出代理髮送的VM的屏幕截圖。 當接收到屏幕截圖時，輸出渲染器將其存儲為本地png圖像文件，並將此文件設置為其SurfaceView組件的源。 SurfaceView組件然後在屏幕上顯示圖像文件。 該過程週期性地，盡可能快地完成，使得用戶將感知應用的屏幕輸出的流，就像應用在本地運行一樣。

input: C. Distributed file system Fig. 4 demonstrates how the components cooperate in the prototype system when an app, installed in the cloud, is downloaded and executed on the smart mobile device. As discussed in Section III-B, we need to share files between the cloud VM and the smart mobile device to allow the app to work properly on the smart mobile device. To achieve this, we implement a distributed file system across the smart mobile device and the cloud in our prototype system.

C.分佈式文件系統 圖4示出了當安裝在雲中的應用在智能移動設備上下載並執行時，組件在原型系統中如何協作。 如第III-B節所述，我們需要在雲VM和智能移動設備之間共享文件，以允許應用程序在智能移動設備上正常工作。 為了實現這一點，我們在我們的原型系統中跨越智能移動設備和雲實施分佈式文件系統。

input: The implementation of the distributed file system contains three parts, as illustrated in Fig. 4. First, the VFS in the Android guest OS is modified such that it communicates with a user-level program depicted as “file system client” in Fig. 4. This is achieved by leveraging the netlink socket mechanism provided by Linux, as Linux is the underlying kernel of Android. Through a netlink socket, a user-level program and the Linux kernel can communicate with each other in a straightforward manner. Second, the file system client is a user-level program running in the Android guest OS on the smart mobile device. This program is implemented as a Linux binary responsible for communicating with both the local Linux kernel and the cloud. Third, the “file system server” in Fig. 4 is a user-level

分佈式文件系統的實現包含三個部分，如圖1所示。 首先，Android客戶操作系統中的VFS被修改，使得其與描述為文件系統客戶端的用戶級程序通信。 。 這是通過利用Linux提供的netlink套接字機制來實現的，因為Linux是Android的底層內核。 通過netlink套接字，用戶級程序和Linux內核可以以直接的方式相互通信。 其次，文件系統客戶端是在智能移動設備上的Android客戶操作系統中運行的用戶級程序。 這個程序被實現為一個Linux二進制，負責與本地Linux內核和雲進行通信。 三，文件系統服務器？ 。 4是用戶級別

input: Fig.4:An app,installedin the cloud,isdownloaded andexecuted on the smart mobile device in the prototype system. Darker blocks indicate the components where our implementation has been involved.

圖4：安裝在雲中的應用程序在原型系統中的智能移動設備上下載並執行。 更深的塊表示我們的實現涉及的組件。

input: program running on the Android-x86 VM in the cloud. This program is also implemented as a Linux binary, responsible for communicating with the file system client on the smart mobile device and executing file operations on the local file system.

程序在雲中的Android-x86 VM上運行。 該程序還實現為Linux二進制，負責與智能移動設備上的文件系統客戶端通信並在本地文件系統上執行文件操作。

input: As illustrated in Fig. 4, the smart mobile device and the cloud communicate with each other through a connection between the file system client and the file system server in the distributed file system. This can also be implemented in several ways, e.g., the VFS on the smart mobile device may directly communicate with the file system server, or even with the VFS on the VM, to avoid switching between the kernel and the user space. We choose to use user-level programs to manage this connection, because they are more robust and easier to configure, even though they introduce some overhead when interacting with the local VFS in the kernel.

如圖1所示。 如圖4所示，智能移動設備和云通過文件系統客戶端和分佈式文件系統中的文件系統服務器之間的連接來彼此通信。 這還可以以若干方式實現，例如，智能移動設備上的VFS可以直接與文件系統服務器通信，或甚至與VM上的VFS通信，以避免在內核和用戶空間之間切換。 我們選擇使用用戶級程序來管理這種連接，因為它們更強大，更容易配置，即使它們在內核中與本地VFS交互時會引入一些開銷。

input: When an app is executed in the cloud, it may only access local files. To access a local file, it will first invoke the system call sys open() to communicate with the VFS, which will open the file and return the file descriptor to the app. Through this file descriptor, the app can read data from the file, write data to the file, or close the file by invoking the sys read(), sys write() and sys close() system calls, respectively. When the app is downloaded from the cloud and executed on the smart mobile device, however, it may access remote files in the cloud. The distributed file system is implemented to support this kind of remote file access in the prototype system.

當應用程序在雲中執行時，它只能訪問本地文件。 要訪問本地文件，它將首先調用系統調用sys open（）與VFS通信，這將打開文件並將文件描述符返回到應用程序。 通過這個文件描述符，應用程序可以分別通過調用sys read（），sys write（）和sys close（）系統調用從文件讀取數據，將數據寫入文件或關閉文件。 然而，當應用從雲中下載並在智能移動設備上執行時，它可以訪問云中的遠程文件。 分佈式文件系統被實現為在原型系統中支持這種遠程文件訪問。

input: To implement this distributed file system, we begin with a simple solution. We redirect every sys open(), sys read(), sys write() and sys close() system call that is invoked on the smart mobile device’s VFS to the VM’s VFS running in the cloud. To be more specific, when an app running on the smart mobile device tries to open a file that is considered in the cloud (e.g., in a directory under /sdcard), the smart mobile device’s VFS will switch to user space by notifying the file system client of this event. Then the file system client will redirect the sys open() system call to the file system server in

要實現這個分佈式文件系統，我們從一個簡單的解決方案開始。 我們將在智能移動設備的VFS上調用的每個sys open（），sys read（），sys write（）和sys close（）系統調用重定向到在雲中運行的VM的VFS。 更具體地，當在智能移動設備上運行的應用嘗試打開在雲中考慮的文件（例如，在下面的目錄

input: 2673

2673

input: ?2015 IEEE Conference on Computer Communications (INFOCOM)

？2015 IEEE計算機通信會議（INFOCOM）

input: the cloud with exactly the same parameters. After receiving this redirected system call, the file system server invokes it on the VM’s VFS and gets the corresponding file descriptor. It then returns this file descriptor to the file system client, which will switch back to the local VFS with the file descriptor it has received. The VFS logs this file descriptor as a remote one and returns it to the app. Then the app can use this file descriptor to read, write, and close the remote file. When the app reads the file through this file descriptor, the VFS will notice that this file descriptor actually refers to a remote file. It will then redirect the sys read() system call to the VM’s VFS similarly to when it redirects the sys open() system call.

雲具有完全相同的參數。 在接收到該重定向系統調用之後，文件系統服務器在VM的VFS上調用它並獲得相應的文件描述符。 然後它將此文件描述符返回到文件系統客戶端，這將切換回本地VFS與它已經接收的文件描述符。 VFS將此文件描述符記錄為遠程文件描述符，並將其返回到應用程序。 然後應用程序可以使用此文件描述符讀取，寫入和關閉遠程文件。 當應用程序通過此文件描述符讀取文件時，VFS會注意到此文件描述符實際上是指遠程文件。 然後它將sys read（）系統調用重定向到VM的VFS，類似於重定向sys open（）系統調用時。

input: The solution described above is straightforward. However, it involves too many network communications between the smart mobile device and the cloud, and too frequent switching between the user space and the kernel on the smart mobile device. As a result, it has prohibitively low performance. To solve this problem, we implemented the distributed file system differently. Fig 4 illustrates part of this new solution, i.e., how it works when the app tries to open a remote file in the cloud. When receiving a sys open() system call from the app and finding that it wants to open a file in the cloud, the smart mobile device’s VFS will switch to user space by notifying the file system client of this event. The file system client then communicates with the file system server, which will send back the data of the corresponding file. The file system client then stores the file on the local disk as a cached file, with a temporary file path that is not likely to conflict with that of any other file. Then the smart mobile device’s VFS opens this file instead of the remote file and returns its file descriptor to the app. The file descriptor is marked as “remote” and the remote file path is recorded in the file descriptor by using two fields that we have added in the file descriptor data structure. Then the app can use this file descriptor to read and write the cached file. When closing the file, the VFS will notice that it is a cached file. It will switch to user space by notifying the file system client of this event. The file system client then sends the cached file back to the file system server, which will overwrite its local file with the cached file in the cloud.

上述解決方案是直接的。然而，它涉及智能移動設備和雲之間的太多網絡通信，以及用戶空間和智能移動設備上的內核之間的頻繁切換。結果，它具有非常低的性能。為了解決這個問題，我們不同地實現了分佈式文件系統。圖4示出了該新解決方案的一部分，即，當應用嘗試在雲中打開遠程文件時，它是如何工作的。當從應用程序接收到sys open（）系統調用並發現它想在雲中打開文件時，智能移動設備的VFS將通過向文件系統客戶端通知此事件來切換到用戶空間。然後文件系統客戶端與文件系統服務器通信，文件系統服務器將發送相應文件的數據。然後，文件系統客戶端將文件作為高速緩存文件存儲在本地磁盤上，臨時文件路徑不可能與任何其他文件的路徑衝突。然後智能移動設備的VFS打開此文件而不是遠程文件，並將其文件描述符返回給應用程序。文件描述符標記為？remote？並且通過使用我們在文件描述符數據結構中添加的兩個字段將遠程文件路徑記錄在文件描述符中。然後應用程序可以使用此文件描述符讀取和寫入緩存的文件。關閉文件時，VFS會注意到它是一個緩存文件。它將通過向文件系統客戶端通知此事件來切換到用戶空間。然後，文件系統客戶端將緩存的文件發送回文件系統服務器，文件系統服務器將使用雲中的緩存文件覆蓋其本地文件。

input: It is arguable that this solution may cause inconsistencywhen an app on the smart mobile device and an app in the cloud access the same file simultaneously. Nevertheless, it is rare that two apps access the same file simultaneously. Most mobile OSes, such as Android, forbid or at least recommend against an app accessing files belonging to another app, for security purposes. Furthermore, although it is not mandatory, a prudent way to use our platform is to create a separate VM in the cloud for every app. Two apps will never access the same file simultaneously in this case.

可以論證的是，當智能移動設備上的應用和雲中的應用同時訪問同一文件時，該解決方案可能導致不一致。 然而，很少有兩個應用程序同時訪問同一個文件。 大多數移動操作系統（如Android）禁止或至少建議應用程序訪問屬於另一個應用程序的文件，以用於安全目的。 此外，雖然它不是強制性的，但使用我們的平台的謹慎方法是在雲中為每個應用程序創建一個單獨的VM。 在這種情況下，兩個應用程序永遠不會同時訪問同一個文件。

input: V. DISCUSSION Hypervisor in local mobile device. To allow for consistency with our system implementation, we present our system design in a way that used a “type 2” hypervisor (i.e., a hosted hypervisor that runs in a host OS) in the local mobile device to achieve high security and privacy guarantees. It is worth

討論本地移動設備中的管理程序。 為了與我們的系統實現保持一致，我們以一種使用類型2的方式呈現我們的系統設計？ 管理程序（即，在主機OS中運行的託管管理程序），以實現高安全性和隱私保證。 值得的

input: noting that it is not necessary to use type 2 hypervisor in our design. Actually, a type 1 hypervisor (i.e., bare-metal hypervisor that runs directly on top of the hardware) fits better into our goal of providing high security guarantees. This is because the main reason we use a hypervisor in local mobile device is to isolate the sensitive inputs (i.e., user taps on the touch screen, data collected by various on-board sensors) from local mobile device OS and apps. These components are greater risks if they are compromised. A type 1 hypervisor, with a small trusted computing base (TCB), can also fulfill our needs, but developing a new hypervisor from the ground up, based on the newly introduced ARM hardware virtualization, would require a lot of effort in engineering optimal parameter settings. Therefore, we opt to use KVM, which is a full-fledge hypervisor that has been recently ported to the ARM architecture, for a fast proof of concept demonstration of our system design. For future work, we plan to develop our own bare-metal hypervisor to further improve our system. The use of Chromebook in the prototype system. We use a Samsung Chromebook as the mobile device in our prototype system implementation. The main reason for this choice is that it has a hardware configuration that supports ARM hardware virtualization, and it requires relatively less effort to enable and run the ARM based KVM, which is the base hypervisor for our prototype system. Although the Samsung Chromebook looks more like a regular laptop, it actually shares the hardware similar to most recent smartphones and tablets. For example, the Chromebook used in our system has a Samsung Exynox 5 Dual SoC, which is the same SoC found in the Google Nexus 10 tablet. The ARM Cortex-A15 MPcore processor contained in the Exynos5 Dual SoC is also used in many other smartphones, like the Samsung Galaxy S4/S5 smartphones, the Galaxy Note 3 smartphone, and the Galaxy Tab Pro tablets. The 2 GB RAM capacity in our Chromebook is also the standard configuration commonly found in the latest smartphones. Therefore, our choice of using Chromebook as the mobile device can fit well into the smart mobile device world in terms of computational capability.

注意在我們的設計中沒有必要使用類型2管理程序。實際上，1型管理程序（即，直接在硬件之上運行的裸機管理程序）更好地適合於提供高安全性保證的目標。這是因為我們使用本地移動設備管理程序的主要原因是隔離敏感輸入（即用戶點擊觸摸屏上，通過各種車載傳感器收集的數據），從當地的移動設備操作系統和應用程序。如果這些組件受損，則這些組件具有更大的風險。 1型管理程序，用一個小的可信計算基（TCB），也能滿足我們的需求，但發展中從地上爬起來的基礎上，新推出的ARM硬件虛擬化一個新的虛擬機管理程序，需要在工程優化參數了很多努力設置。因此，我們選擇使用KVM，這是一個功能完善的虛擬機管理程序最近已移植到ARM架構，我們的系統設計概念論證的快速證明。對於未來的工作，我們計劃開發自己的裸機管理程序，以進一步完善我們的系統。在原型系統中使用Chromebook。在我們的原型系統實施中，我們使用三星Chromebook作為移動設備。這樣選擇的主要原因是，它有一個支持ARM硬件的虛擬化的硬件結構，它需要相對較少的努力，以使和運行基於ARM的KVM，它是我們的原型系統的基地管理程序。雖然三星Chromebook看起來更像一台普通的筆記本電腦，但它實際上共享的硬件類似於最新的智能手機和平板電腦。例如，我們系統中使用的Chromebook具有三星Exynox 5雙SoC，與Google Nexus 10平板電腦中的SoC相同。 Exynos5雙SoC中包含的ARM Cortex-A15 MPcore處理器也用於許多其他智能手機，如三星Galaxy S4

input: VI. EVALUATION In this section, we describe the real-world experiments we have conducted to evaluate the performance of our prototype system.

VI。 評估在本節中，我們描述了我們為評估我們的原型系統的性能而進行的真實世界實驗。

input: A. Experimental setup Our prototype system consists of two parts, the smart mobile device and the cloud VM. The smart mobile device is a Samsung Chromebook featuring the Exynos 5 Dual SoC, 2 GB RAM and 16 GB SSD hard drive. The Exynos 5 Dual SoC is equipped with a 1.7 GHz dual-core ARM Cortex-A15 processor. The ARM Cortex-A15 processor has hardware virtualization support, which allows us to implement the proposed hypervisor in our prototype system. The Chromebook runs Linux (kernel version 3.13.0) as the host OS. On top of the hypervisor, we run Android Jelly Bean (Android version 4.1.1, Linux kernel

A.實驗設置我們的原型系統由兩部分組成，智能移動設備和雲VM。 智能移動設備是採用Exynos 5雙SoC，2 GB RAM和16 GB SSD硬盤驅動器的三星Chromebook。 Exynos 5雙SoC配備了1.7 GHz雙核ARM Cortex-A15處理器。 ARM Cortex-A15處理器具有硬件虛擬化支持，允許我們在我們的原型系統中實現提出的虛擬機管理程序。 Chromebook運行Linux（內核版本3.13.0）作為主機操作系統。 在管理程序之上，我們運行Android Jelly Bean（Android版本4.1.1，Linux內核

input: 2674

2674

input: ?2015 IEEE Conference on Computer Communications (INFOCOM)

？2015 IEEE計算機通信會議（INFOCOM）

input: Response time (millisecond)

響應時間（毫秒）

input: 25000

25000

input: 20000

20000

input: 15000

15000

input: 10000

10000

input: 5000

5000

input: 0 0.25 14 KB 82 KB 178 KB 236 KB 392 KB 512 KB 0.5 1 2 10 100 Network bandwidth (Mb/s)

0 0.25 14 KB 82 KB 178 KB 236 KB 392 KB 512 KB 0.5 1 2 10 100網絡帶寬（Mb

input: Fig. 5: App response time if running in the cloud: UI response time (Y-axis) of screenshots with different size under different network conditions (X-axis).

圖。 5：在雲中運行時的應用響應時間：在不同網絡條件（X軸）下具有不同大小的截屏的UI響應時間（Y軸）。

input: version 3.9.0) as the guest OS. The cloud VM runs Androidx86 [25] as the guest OS (Android version 4.3, Linux kernel version 3.10.2). It is hosted by the VMware workstation 10.0.1 virtual machine monitor (VMM) running on a host PC with an Intel Core i7 CPU (2.3 GHz) and 8 GB RAM. The smart mobile device and the cloud VM are connected through an 802.11n wireless router.

版本3.9.0）作為客戶機操作系統。 雲VM運行Androidx86 [25]作為客戶操作系統（Android版本4.3，Linux內核版本3.10.2）。 它由運行在具有Intel Core i7 CPU（2.3 GHz）和8 GB RAM的主機PC上的VMware工作站10.0.1虛擬機監視器（VMM）託管。 智能移動設備和雲VM通過802.11n無線路由器連接。

input: B. Running apps in the cloud: the response time The most significant feature of our system is that we allow the entire mobile device OS and the apps to run in the cloud. Therefore, we first evaluate the app response time when the apps are running in the cloud. In our system, we send the output of an app running in the cloud back to the device by first taking screenshots of the app, and then transmitting them back to the device. Therefore, there are two major factors that can affect app response time, network bandwidth and the size of each screenshot. We design an experiment to evaluate the impacts of thesetwofactors.In ourexperiment,we usean imageviewer app to open different pictures. Remember that with our system, the app launch and the picture opening operations are triggered by the user on the mobile device side, and the actual operations are conducted on the cloud side. Once a picture is opened, our system takes a screenshot of it, and sends the screenshot back to the mobile device to display to the user. We measure the response time as the time difference between the point when the user triggers the picture opening action on the mobile device, and the point when the screenshot is sent back from the cloud and displayed. We choose different pictures, such that we have screenshots with different sizes (14 KB, 82 KB, 178 KB, 236 KB, 392 KB, and 512 KB are used in our experiment). We also configure the wireless router to achieve different network bandwidth rates (0.25 Mb/s, 0.5 Mb/s,1 Mb/s,2 Mb/s, 10 Mb/s, and 100 Mb/s are used in our experiment).

B.在雲中運行應用程序：響應時間我們的系統最重要的功能是，我們允許整個移動設備操作系統和應用程序在雲中運行。因此，我們首先評估應用在雲中運行時的應用響應時間。在我們的系統中，我們通過首先截取應用程序的屏幕截圖，然後將它們發送回設備，將運行在雲中的應用程序的輸出發送回設備。因此，有兩個主要因素可以影響應用程序響應時間，網絡帶寬和每個屏幕截圖的大小。我們設計了一個實驗來評估這些效應的影響。在實驗中，我們使用一個imageviewer應用程序打開不同的圖片。記住，在我們的系統中，應用程序啟動和圖片打開操作由用戶在移動設備側觸發，並且實際操作在雲端進行。一旦圖片打開，我們的系統會截圖，並將屏幕截圖發送回移動設備以顯示給用戶。我們將響應時間測量為用戶在移動設備上觸發圖片打開動作的時間點與從雲發送回並且顯示屏幕截圖的點之間的時間差。我們選擇不同的圖片，使我們有不同大小的屏幕截圖（14 KB，82 KB，178 KB，236 KB，392 KB和512 KB在我們的實驗中使用）。我們還配置無線路由器以實現不同的網絡帶寬速率（0.25 Mb

input: Fig. 5 depicts the experiment results. We can see that network bandwidth has a significant impact on app response time when the bandwidth is small. But this impact gradually

圖。 圖5描述了實驗結果。 我們可以看到，當帶寬較小時，網絡帶寬對應用程序響應時間有重大影響。 但這種影響逐漸

input: TABLE I: Hypervisor overhead

表I：管理程序開銷

input: Native OS With unmodified hypervisor With our hypervisor Delay 4 ms 63 ms 65 ms

本地操作系統帶有未修改的虛擬機管理程序使用我們的虛擬機管理程序延遲4毫秒63毫秒65毫秒

input: diminishes as the available bandwidth increases. For example, when the network bandwidth is fixed at 0.25 Mb/s, it takes only 0.5 second to open a picture 14 KB in size. But it needs almost 24 seconds to get back a 512 KB screenshot. The

隨著可用帶寬的增加而減小。 例如，當網絡帶寬固定為0.25 Mb時

input: 512KB

512KB

input: response time ratio for these two screenshot sizes (i.e., )

響應時間比率，這兩個屏幕截圖尺寸（即，）

input: 14KB

14KB

input: 24second

24秒

input: is = 48. But when the network bandwidth is set at 100

是= 48.但是當網絡帶寬設置為100時

input: 0.5second

0.5秒

input: Mb/s, the response time ratio for the same screenshot sizes (i.e.,

Mb

input: 512KB 0.4second

512KB 0.4秒

input: ) is only =5.7. Fortunately, modern mobile

）只有= 5.7。 幸運的是，現代移動

input: 14KB 0.07second

14KB 0.07秒

input: data networks can support a very high transmission rate. The 4G LTE network has a peak download speed approaching 100 Mb/s [26], and on a typical day 4G download speeds can range from 2.8 Mb/s to 9.1 Mb/s, with an average value of 6.2 Mb/s [27]. With the average 4G download speed (6.2 Mb/s), the app response time of our prototype system for a screenshot of 512 KB is only about one second.

數據網絡可以支持非常高的傳輸速率。 4G LTE網絡的峰值下載速度接近100 Mb

input: C. Performance of the hypervisor on the local device Our system exploits the ARM hardware virtualization technology to achieve user/sensor input isolation from the mobile device OS. Specifically, when an app is running in the cloud, our hypervisor intercepts all the inputs from the user (e.g., touch screen,keyboard) and sensors, performs an encryption on them, and sends them to the cloud. We design an experiment to evaluate the overhead introduced by our hypervisor. In this experiment, we use keyboard input to evaluate the hypervisor overhead. We test three cases. In the first case, we use the keyboard to provide inputs for a user program running on the native OS of the mobile device. In the second case, we test the same scenario, except that the user program is running in the guest OS of the mobile device. In this case, the unmodified hypervisor will introduce some overhead to the keyboard input operation. The third test case shares the same setup as the second one, except that the hypervisor is the one used in our prototype system, which performs encryption on the intercepted keyboard inputs. Because we want to test the overhead caused by our hypervisor, we direct the encrypted input up to the user program in the guest OS, instead of redirecting it to the cloud. In all the three cases, we measure the time delay between the keyboard input interrupt and the time when the user program receives the input.We perform each test ten times, and report the average value here. Table I shows the results of the experiment. The results suggest that by running the user program in the guest OS, the time delay increases by one order of magnitude, compared to that when the user program is running in the guest OS. This is normal because the guest OS involves many switches between many entities including the guest OS, the hypervisor, the host OS, and the QEMU hardware emulator (required by KVM). It is worth noting that when comparing to the case with the unmodified hypervisor, our hypervisor only incurs a very small amount of additional delay (about 3%).

C.本地設備上的管理程序的性能我們的系統利用ARM硬件虛擬化技術來實現用戶

input: 2675

2675

input: ?2015 IEEE Conference on Computer Communications (INFOCOM)

？2015 IEEE計算機通信會議（INFOCOM）

input: TABLE II: Performance of the distributed file system

表II：分佈式文件系統的性能

input: 64 KB 256 KB 512 KB 1 MB 5 MB 10 MB Open 36 ms 85 ms 115 ms 225 ms 827 ms 1324 ms Read 1 ms 1 ms 5 ms 11 ms 20 ms 38 ms Write 1 ms 3 ms 6 ms 13 ms 68 ms 128 ms Close 39 ms 68 ms 116 ms 219 ms 680 ms 1337 ms

64 KB 256 KB 512 KB 1 MB 5 MB 10 MB打開36 ms 85 ms 115 ms 225 ms 827 ms 1324 ms讀取1 ms 1 ms 5 ms 11 ms 20 ms 38 ms寫入1 ms 3 ms 6 ms 13 ms 68 ms 128 ms關閉39 ms 68 ms 116 ms 219 ms 680 ms 1337 ms

input: TABLE III: Performance of the native file system

表III：本地文件系統的性能

input: 64 KB 256 KB 512 KB 1 MB 5 MB 10 MB Open 1 ms 1 ms 2 ms 5 ms 14 ms 24 ms Read 1 ms 1 ms 6 ms 11 ms 21 ms 40 ms Write 1 ms 3 ms 6 ms 13 ms 67 ms 126 ms Close 1 ms 1 ms 2 ms 3 ms 14 ms 27 ms

64 KB 256 KB 512 KB 1 MB 5 MB 10 MB打開1 ms 1 ms 2 ms 5 ms 14 ms 24 ms讀取1 ms 1 ms 6 ms 11 ms 21 ms 40 ms寫入1 ms 3 ms 6 ms 13 ms 67 ms 126 ms關閉1 ms 1 ms 2 ms 3 ms 14 ms 27 ms

input: D. Performance of the distributed file system The purpose of our distributed file system is to allow users to run apps locally on their mobile device. In this experiment, we evaluate the performance of file open, read, write, and close operations of our distributed file system. Each test is performed ten times. Table II shows the performance results of our distributed file system. As a comparison, Table III shows the results of the native file system. From these results we can see that our file system incurs a time overhead for the open and close operations. This is because when an app running in the mobile device tries to open a certain file that is not available in the local device, our file system transparently caches the file of interest from the cloud to the mobile device to allow the app to proceed. When the app finishes accessing and closes the file, our file system automatically writes the file back to the cloud. Therefore opening files is costly. But writing and reading them is much less so. Since we are using whole file caching in our current implementation, file size and network bandwidth have major impacts on the open/close delay.For file read/write, since our file system allows local access to the cached copy, it has the same performance as the read/write operations in a native file system.

D.分佈式文件系統的性能我們的分佈式文件系統的目的是允許用戶在其移動設備上本地運行應用程序。在這個實驗中，我們評估我們的分佈式文件系統的文件打開，讀取，寫入和關閉操作的性能。每次測試進行10次。表II顯示了我們的分佈式文件系統的性能結果。作為比較，表III示出了本地文件系統的結果。從這些結果，我們可以看到，我們的文件系統導致開放和關閉操作的時間開銷。這是因為當在移動設備中運行的應用嘗試打開在本地設備中不可用的某個文件時，我們的文件系統將感興趣的文件從雲緩存到移動設備以允許應用繼續進行。當應用程序完成訪問和關閉文件時，我們的文件系統會自動將文件寫回雲。因此打開文件是昂貴的。但是寫作和閱讀它們並不是那麼容易。由於我們在當前的實現中使用整個文件緩存，文件大小和網絡帶寬對開放性有重大影響

input: VII. CONCLUSION The mobile-cloud computing model will be the dominating trend in the future. However, existing solutions do not fully exploit the potential of this model. To this end, we aim at designing a solution that goes beyond the current state of the art. In this paper, we propose a novel mobile-cloud platform with two fundamental contributions. First, our platform allows users to freely choose to run their applications either in the cloud or on their local devices. We feel that this is a very useful and practical feature for users, and believe we are the first to consider this situation in a mobile-cloud platform of this kind. Second, our platform provides security guarantees against untrusted applications and an untrusted local device’s operating system, by leveraging hardware virtualization technology. To the best of our knowledge, we are the first to utilize hardware virtualization to strengthen the security on mobile devices.

VII。 結論移動雲計算模型將是未來的主導趨勢。 然而，現有的解決方案沒有充分利用這個模型的潛力。 為此，我們的目標是設計超越現有技術水平的解決方案。 在本文中，我們提出了一個具有兩個基本貢獻的新型移動雲平台。 首先，我們的平台允許用戶自由選擇在雲中或在其本地設備上運行他們的應用程序。 我們認為這是一個非常有用和實用的功能，用戶，並相信我們是第一個考慮這種情況在這種移動雲平台。 第二，我們的平台通過利用硬件虛擬化技術為不受信任的應用程序和不受信任的本地設備操作系統提供安全保證。 據我們所知，我們率先利用硬件虛擬化來加強移動設備的安全性。

input: Based on these design concepts, we build a prototype system on a Chromebook acting as the user’s local mobile device, and a commodity x86 laptop PC, which acts as a cloud server. Our evaluation on the prototype system proves that our platform is useful and pragmatic.

基於這些設計理念，我們在作為用戶本地移動設備的Chromebook和作為雲服務器的商用x86筆記本電腦上構建了一個原型系統。 我們對原型系統的評估證明我們的平台是有用和務實的。

input: ACKNOWLEDGMENT The authors would like to thank all the reviewers for their helpful comments. This project was supported in part by US National Science Foundation grants CNS-1320453 and CNS1117412.

致謝作者感謝所有審稿人的有益意見。 該項目部分得到美國國家科學基金會資助CNS-1320453和CNS1117412的支持。

input: REFERENCES

參考文獻