VirtSense: Virtualize Sensing through ARM TrustZone on Internet-of-Things

Renju Liu
UCLA CS
rl@cs.ucla.edu

Mani Srivastava
UCLA EE
mbs@ucla.edu

ABSTRACT

Internet-of-Things (IoTs) are becoming more and more popular in our life. IoT devices are generally designed for sensing or actuation purposes. However, the current sensing system on IoT devices lacks the understanding of sensing needs, which diminishes the sensing flexibility, isolation, and security when multiple sensing applications need to use sensor resources. In this work, we propose *VirtSense* , an ARM TrustZone based virtual sensing system, to provide each sensing application a virtual sensor instance, which further enables a safe, flexible and isolated sensing environment on the IoT devices. Our preliminary results show that *VirtSense* : 1) can provide virtual sensor instance for each sensing application so that the sensing needs of each application will be satisfied without affecting others; 2) is able to enforce access control policy even under an untrusted environment.

CCS CONCEPTS

• **Computer systems organization → Sensors and actuators**; *Sensor networks*; • **Hardware → Sensor devices and platforms**; *Sensor applications and deployments*; • **Software and its engineering → Virtual machines**; **Virtual worlds software**; **Embedded software**; • **Security and privacy →** *Access control*; *Authorization*;

ACM Reference Format:
Renju Liu and Mani Srivastava. 2018. VirtSense: Virtualize Sensing through ARM TrustZone on Internet-of-Things. In *3rd Workshop on System Software for Trusted Execution (SysTEX '18), October 15, 2018, Toronto, ON, Canada.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3268935.3268937

1 INTRODUCTION

The number of Internet-of-Things (IoTs) has dramatically increased in the past few years. Unlike the early stage IoT devices that are mainly designed as sensor extensions, the current IoT devices are no longer only for data collecting but are capable of learning, or in other words, smart sensing [15]. The key transition of the software system from "sensor extensions" to "smart sensing" is the changeover from single-tenancy sensing system to multi-tenancy sensing system. We refer single-tenancy sensing system as the system where only one application can access the physical sensors, and the multi-tenancy sensing system as the system that multiple applications can access the same sensors simultaneously without interfering with each other.

The current sensing system on IoT devices inherits the concepts from mobile systems, Linux-based systems, or embedded systems. Such migration shows an inefficiency while workloads have been changed [18, 19] from data collecting to sensing. The sensing services on Linux or embedded systems is conceptually analogous to single-tenancy sensing system because when an application needs to use sensor resources, it sets up a mutex lock so that other applications cannot modify it if a different sampling rate is sought. These sensing mechanisms fail to meet the fundamental requirement of multi-tenancy sensing system because the simultaneous sensor access from multiple applications with different sensing rate is prohibited. On the mobile system such as Android, the sensing service is performed through a *max-for-all* mechanism. The OS only provides the data with maximum sampling rate from all applications although most applications do not need such a sensing granularity. On iOS, the sensor management simply discards the excessive data when doing a downsampling for different apps. Moreover, every application only allows having one sensing rate on iOS, so all threads in a multiple-threaded application are only allowed to sense at one rate. Although the sensing service from the mobile system is more advanced than embedded system's sensing service, all of the burdens of dealing with excessive sensor data fall on the applications themselves. Nevertheless, the sensing system should take over the burdens to achieve the sensing isolation among and within applications on the multi-tenancy system. Moreover, the application workloads on mobile devices are different from those on IoT devices, where IoT devices are mostly used for sensing or actuating, but mobile devices lean more as user-interactive workloads. Hence, optimizing the sensing system on mobile devices is not an easy implementation.

Another challenge faced by multi-tenancy sensing system is the enforcement of sensor access control policy. The traditional access control mechanism on IoT devices inherited from mobile systems usually seeks for the user's permission at the time when an application launches. Under this access control mechanism, if the state of the device changes while using the application, and the new state no longer permits the access to certain sensors, the access control policy will not be able to execute such permission changes at runtime. Some research work [6] tracks the data flow to enforce access control policy at runtime. However, these access control mechanisms rely on the OS boundary of user space and kernel space. If the OS gets compromised, all these access control mechanisms can be bypassed.

To remedy the shortcomings of current sensor systems for IoT devices, we propose *VirtSense* as illustrated in figure 1, an enclave

SysTEX '18, October 15, 2018, Toronto, ON, Canada
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-5998-6/18/10…$15.00
https://doi.org/10.1145/3268935.3268937

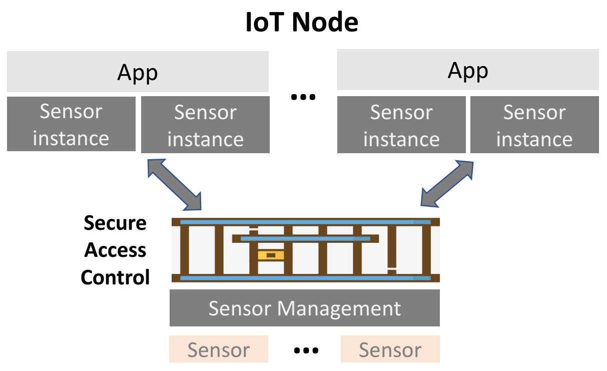

Figure 1: VirtSense Architecture

and virtualization-based framework that meets the real-time demands of multi-tenancy sensing system. The goals of such a system are to achieve the sensing isolation among different sensing applications and to enforce the access rules in an untrusted environment. These goals enable the simultaneous and legit access to sensors from different applications without affecting each other.

The concept of virtualization helps the sensing system multiplex the limited physical sensor resources to different sensing applications through sensing resampling techniques. While software-based virtualization introduces extra overheads, we balance the trade-offs between sensing accuracy and data delivery speeds. We expose such trade-offs as options for developers to choose so that they can control the granularity of sensor accuracy and data delivery speed.

Enclave technology furnishes *VirtSense* with a trusted computing base (TCB), which further guarantees the enforcement of access control policy. The enclave TCB will maintain its integrity when the OS gets compromised because of the hardware protection. One of the design consideration for TCB is to keep its minimal size due to the extra execution overhead. Hence the balance of how often in enforcing permission rules should be carefully chosen. If the access permission is checked too frequently, the overhead is too large; if the access permission is checked too rarely, it will not provide access protection. In *VirtSense* , we allow the developers to choose how often they want their access rules to be enforced, with respect to the access request from the applications.

Our implementation of *VirtSense* uses ARM TrustZone based devices and prototypes the architecture on Raspberry Pi. Our preliminary results show that *VirtSense* satisfies multi-tenancy sensing needs by: 1) Allowing each sensing application control over its specific set of sensing requirements; 2) the access control policy is enforced under simulated compromised OS.

2 BACKGROUND

2.1 Sensing System

Most embedded systems and Linux-based OSes only allow one application to sense at a given rate, while all other applications have to wait until the one releases the resource if they have different sensing needs. On Android, the sensor manager manages sensing requests. All the applications send the sensing requests to the sensor manager through a Binder message, at which the sensor manager will then obtain the sensor data and distribute it to the applications. The sensing rate is not guaranteed because the sensor manager chooses the largest sensing rate from all the applications and broadcast the data to all sensing applications.

2.2 ARM TrustZone

ARM TrustZone is the enclave technology originally developed by ARM company. It is implemented on all A-series processes and two m-series (m23 and m33). ARM TrustZone separates the execution environment into two - normal world and secure world. The normal world is the untrusted environment running an untrusted OS. The secure world is the trusted computing base of TrustZone, which hosts trusted environment. The context switch between the normal world and secure world is done through Secure Monitor Call (SMC). The code that runs in the secure world has higher privilege than the code in the normal world, and the secure world is able to define the memory region that can only be accessed by privileged code. These regions could be regular memory or memory mapped registers of peripherals. If the code running in the normal world tries to access the protected memory regions, TrustZone throws a hardware exception.

3 DESIGN OF VIRTSENSE

Our key motivation of *VirtSense* sensing system on IoT devices is that the sensing system should take care of the sensing needs from different sensing applications. These sensing needs are the sensing requests for periodic sensing at different sampling rate or event-driven sensing requests. Under the current sensing framework such as Android or iOS, the burden of taking care of the sensing difference is on the applications' side. In other words, the current sensing framework lacks the understanding of applications' sensing needs. Moreover, the current sensor access control mechanisms purely relies on the isolation and protection from the OS. When an OS gets compromised by a malicious application containing rootkit, the access control mechanisms will no longer protect the device.

We provide a sensing framework *VirtSense* as illustrated in figure 1 that supports the discrepancy of multiple application sensing workloads with the emphasis on sensing flexibility, accuracy and security. In this section, we will discuss the design of the framework.

3.1 Design Goals

To overcome the shortcomings of the current sensing framework while not diminishing the functionalities of the existing sensing framework, we set up the following goals for the design of our new sensing framework.

- **Sensing simplicity.** Current sensing framework pushes the sensing burdens to each application. If one application changes its sampling rate, it might affect all other applications that are using the same sensing service. Nevertheless, while dealing with the different sensing requests from different applications, the sensing framework needs to manage the difference of the sampling requests but not the applications themselves. Due to the hardware resource limitation that each physical sensor can only have one sampling rate, the sensing framework should multiplex the physical sensor values to different sensing events. The sensing system should provide a high-level abstract for multi-tenancy sensing applications but not pass this burden to the applications.

- **Options for balancing sensing accuracy and sensor data delivery speed.** Downsampling or upsampling are needed when the sensing framework multiplexes the physical sensor values to each sensing request at their requested sensing rate. Downsampling or upsampling require data reconstruction, and different techniques will affect the data accuracy and the delivering speed because higher accuracy requires more data to reconstruct the sensing signal, which lowers the data delivering speed. The abstraction of choosing such downsampling or upsampling sensing techniques should be exposed to the application developers so that they can properly determine the most beneficial downsampling or upsampling techniques for their applications.
- **Sensor access security.** All the existing sensor access control mechanisms, such as asking users for permissions when launching an app and use data-flow to track the access control, rely on the security of user space and kernel space boundary. While designing the new sensing framework, we aim at the access security of sensors even if the OS is compromised (i.e. a broken of user and kernel space). The access control policy needs to be independent of applications but only controls the behaviors how an application accesses the sensory data. Furthermore, the access control should have the capability of dynamically adjusting sensor access permission while using the device rather than only asking for the permission at the application launching time. For example, an access control rule can reject the microphone sensing request if the device is in a secret conference room, while approving it if the device is outside a conference room.
- **Scalability.** The sensing framework needs to have a proper level of scalability. When new software or hardware sensors are added to the system, the new sensing framework is able to integrate them to the existing system with the sensors' driver modules without overhauling the whole sensing system.

3.2 Design Principles

To achieve the goals explained in section 3.1, we present *VirtSense* sensing framework that virtualizes physical sensors based on enclave technology. While designing *VirtSense* , we set forth two essential principles *VirtSense* needs to follow with.

3.2.1 Principle of Various Sensing Workloads. Every sensing application has its own requirements of sensing work. The examples of the sensing workloads can be different sensing rate for periodical sensing or different event-driven sensing requests. The various sensing workloads also indicate that different applications need to share the limited physical sensing resources. *VirtSense* needs to understand the discrepancies among all the sampling requests and satisfy these needs.

Embracing sensor virtualization in VirtSense . Virtualization overrides the hardware resource limitations by allowing a software management to multiplex the hardware resources for each virtual instance, which allows applications to request one or more instances for the resources while each instance is independent and

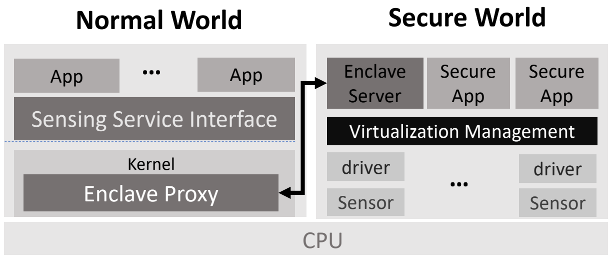

Figure 2: VirtSense Overview

does not interfere with each other. Specifically speaking, the virtual sensor instance in *VirtSense* refers to an instance for a physical sensor or a combination of multiple physical sensors (e.g. software sensors). Each application no longer needs to be "penalized" while other applications require different sensing requests. For example, when application A requests a periodic sampling with sampling rate of 60 Hz while application B requests a periodic sampling with 100 Hz and application C requests an event-driven sampling, the application A, B and C themselves do not need to comprehend the difference of other applications since *VirtSense* is responsible for multiplexing the physical sensor resources. Meanwhile, each application can also choose the sensing accuracy for their instances. By virtualizing the physical sensors, *VirtSense* is committed to provide applications `sensing simplicity`, `scalability` and the options for `sensing accuracy`.

3.2.2 Principle of Untrusted OS Execution Runtime. Due to the large size and various functionalities of the OS, OS can get compromised through malicious kernel extensions or rootkits. Hence, the compromised OS have the abilities to bypass all the security protection mechanism. Under *VirtSense* , it must ensure the confidentiality and integrity of its code especially access control code, which implies that the code is executed under a possible compromised OS environment. Enclave technology provides a protection infrastructure to allow *VirtSense* to run securely under an untrusted environment.

Embracing enclave in VirtSense . Enclave provides a hardware level protection for the confidentiality of essential memory-mapped peripherals. Using the enclave protection can guarantee the required safety policy to be enforced regardless of the health status of the OS. The enclave will throw a hardware exception if the malicious OS tries to access the protected memory regions including those memory-mapped registers for peripherals without passing through the access control policy. Differing from the traditional OS protection boundary for the access control of all sensors, *VirtSense* adopted enclave for the access for the sake of an enhanced security, which provides the system an improved `sensor access security`. Enclave in *VirtSense* provides a minimal yet sufficient trusted computing base. The access for sensors from the OS outside the enclave is prohibited, and the only method to use the sensor resources from the OS is to send an access petition to TCB in the enclave. This protection layer makes sure the sensor access policy inside TCB is enforced regardless of the security status of the OS.

4 *VIRTSENSE* PROTOTYPE

We prototype *VirtSense* with four key components that are sensing service interface, enclave proxy, secure apps, and virtualization management as shown in figure 2. These four parts provide a secure interface and sensing service for the applications in the normal world.

4.1 Threat Model

Under *VirtSense* framework, we assume that the applications in the normal world are not trusted. These applications could be intentionally malicious and compromise the whole OS. They can be downloaded and installed from any third party manufacturers while the applications and execution runtime inside secure world are secure and trusted. The secure world applications need to be conducted extra scrutinizing by an authorized party.

Sensing service interface. Sensing service interface provides the interface APIs to applications for sensing instances creation and the sensing rate and accuracy manipulation. To keep a proper high-level abstraction, sensing service interface is the only exposure point between the applications and *VirtSense* service. It also plays a role of load balancer of *VirtSense* , which determines how to deliver the sensor values to each application with minimal overhead. For example, if all applications sample at 60 Hz, the sensing service interface will only create one instance for all the applications even if every application requires a virtual sensing instance.

Enclave Proxy. Enclave Proxy communicates with the secure world inside the enclave. It batches and sends the sensing requests from the normal world applications to the enclave server in the secure world through SMC driver. Moreover, the sensor data sent from secure world are also received by the enclave proxy that further delivers to sensing service interface and applications.

Enclave Server. Enclave server is the communication midpoint between the normal world and the secure world, residing in the secure world. Enclave server processes the sensing request batches from the normal world applications and send them to the virtualization management. It also packs up the sensor data updated from the virtualization management and delivers them back to the applications in the normal world. Enclave proxy and enclave server are the two communication endpoints that bridge the sensing requests sent from the normal world and the sensor data delivered back from the secure world.

Secure Apps. Secure apps, which are the key components of access control mechanism, are pre-installed by the users. They are used to control the access from normal world applications to sensors. All the sensor access requests need to be sent to secure apps through virtualization management for access authenticity check. Secure apps can contain rules defined by the users to restrict the access of sensors. For example, a rule can be depicted as when the device is within a certain room, the speaker cannot sense the environment by the applications. If the decision is more complicated to make than using rules, a pre-trained neural network can be integrated with the secure apps. Moreover, because secure apps have no restrictions to access the protected sensors, the pre-trained neural network is capable of being further trained while using the device. The sensor access request from normal world applications can only be granted if all secure apps approve such access.

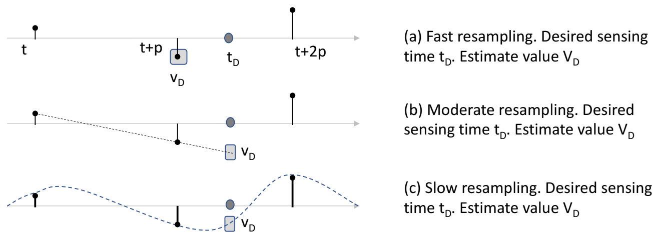
t
t+p
V_D
t_D
t+2p
(a) Fast resampling. Desired sensing time t_D. Estimate value V_D
V_D
(b) Moderate resampling. Desired sensing time t_D. Estimate value V_D
V_D
(c) Slow resampling. Desired sensing time t_D. Estimate value V_D

Figure 3: Example of resampling methods. Physical sensor value updated at t, t+p and t+2p, where p is the physical sensor sampling rate. t_D is the desired sensing time for a virtual sampling request, and V_D is the returned value marked in a grey box from *VirtSense* .

Virtualization management. Virtualization management enforces the access policies from secure apps and dynamically chooses the sensing sampling rate set for the real physical sensors to satisfy the sensing requests from the normal world applications. virtualization management receives the sensing requests through enclave server, and it checks whether it has violated the access rules from either of the secure apps. If the access request is permitted, the virtualization management will change the sensing sampling rate if necessary and send sensor data back to enclave server when they are updated. The details of the virtual sensing algorithms and APIs provided to the developers will be introduced in section 4.2.

4.2 Sensing Virtualization Algorithm

The algorithm of *VirtSense* virtualization management is to provide each sensing instance a resampled sensor value calculated based on the physical sensor value. As we discussed in the previous section, *VirtSense* allows both event-driven sensing and periodic sensing. In *VirtSense* , each application is permitted to create one or more sensing instance. For different sensing instance, the application developers can specify a resampling method to satisfy the sensing needs. *VirtSense* sets the maximum sensing rate among the sampling requests by all the applications as the sampling rate for the physical sensor. Figure 3 shows a demonstration of different sensing resampling techniques. All the raw data for the resampling are pre-processed by the sensor's firmware.

Fast resampling. Fast resampling method returns the closest available sensor value at the requested sampling time. This method provides sensor resampling values at with a low accuracy but high speed. When the application developer chooses this method, *VirtSense* will deliver the sensing value at the highest priority regarding the other two resampling methods.

Moderate resampling. Moderate resampling uses a linear approximation strategy to resample the sensor values. This method will provide a more accurate sensor resampling value than fast resampling method while the calculation and delivering speed will be slower than that. *VirtSense* assigns moderate resampling a medium priority while delivering the sampling values to the applications.

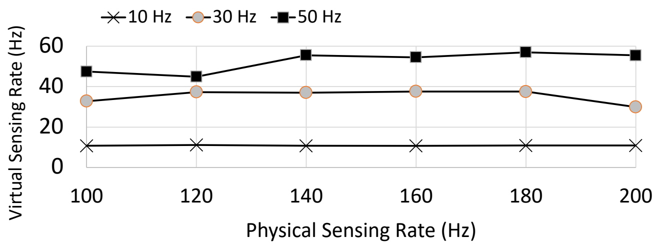

Figure 4: Running application with desired sampling rate 10Hz, 30Hz and 50Hz simultaneously (fast resampling) with physical sensor value from 100Hz to 200Hz with 20Hz increment in *VirtSense* .

Slow resampling. Slow resampling is supposed to provide the most accurate resampled data. These sample data recovering techniques are high-order approximation with least square estimation or [2, 9, 10]. Using these methods will be given the lowest priority hence lowest delivering speed while delivering the data to the applications.

4.3 Sensor Security and Access Control

All the access to the sensors is restricted by the secure world. The secure world prohibits the direct access to the sensors from the normal world OS. But rather, the normal world applications need to send the access request to TCB residing in the secure world. In *VirtSense* , we implement secure apps as the safety rule checkers to check the validity of the access request. When enclave server receives the batched access requests from the enclave proxy, it passes the requests to secure apps through virtualization management, and then the secure apps will determine whether the access is allowed. Because the context switch between the normal world and the secure world is expensive, secure apps cannot track the information flow of how sensor data objects are used such as TaintDroid [6].

5 EVALUATION

5.1 Experiment Setup

Our experiment uses Raspberry Pi 3 as the platform. It is one of the cheapest ARM A-series development board. We use Open-Portable Trusted Execution Environment (OP-TEE)[1] for the software setup. OP-TEE is an ARM TrustZone enabled prototype operating system. OP-TEE uses Linaro Linux in the normal world and Rich OS in the secure world based on GlobalPlatform TEE Client API. We also generate simulated sensor data at different sensing rate. Because the secure world runtime of OP-TEE does not have inter-process communication based on the message queue, we use a secure-world applications shared storage file to emulate the inter-process communication.

To evaluate our preliminary implementation, we launch three periodic sampling applications with sampling rate at 10Hz, 30Hz and 50Hz, and several other malicious applications with various sampling rate. We also provide a secure app that contains the rule to only allow sensor access if they are sampled at 10Hz, 30Hz or 50Hz. To compensate for the overhead introduced by the system, the real sampling rate is adjusted to its expected values. We measure the average sampling received for each application as shown in figure 4.

[1]https://www.op-tee.org/

5.2 Preliminary Results

Overhead. Our results show that the overhead introduced by *VirtSense* is 16ms on average. The overhead is calculated by two clocks. The first clock starts before calling into secure world and ends up with exiting from the secure world. The second clock measures the overhead introduced by the access control applications inside the secure world. By further breaking down the overhead, roughly 0.18 ms comes from the context switch between the normal world and the secure world, and the rest comes from the enforcement of security rules by security apps. The current security rule enforcement communicates to sensor virtualization management through a shared secure file, and it can be further optimized through a more efficient inter-process communication such as message queuing.

Multi-tenancy sensing virtualization. In order to test how physical sensing rate could possibly affect virtual sensing rate, we run three sensing applications requiring 10Hz, 30Hz and 50Hz sensing rate at the same time with different physical sensor rate as shown in figure 4. We let the applications run for about ten seconds and calculate the average sampling rate. The average sampling rate shows that when the desired sampling rate is low, the more accurate the frequency it receives the data at the expected rate.

Security analysis. We install a secure app that restricts the access of all the sensing requests except with sensing rate 10Hz, 30Hz or 50Hz. The secure app successfully blocked the sensing requests not specified by the rule. Our implementation of *VirtSense* does not provide protection against DDoS attack because each request must be scrutinized by secure apps. In this case, DDoS attack mainly refers to spoofing the sensors or tampering the actuators from software level. If we carefully develop a secure app that restricts a certain number of access requests in a unit time, it could be robust to DDoS attack as well.

5.3 Future Work

VirtSense provides virtualization of sensors, while in reality, sensors are also associated with actuators. Hence, a possible future work is to explore how we can relate actuation and sensing together as a virtual control system.

6 RELATED WORK

Enclave technology has been widely deployed in all kinds of different systems recent years. For example, Intel SGX is for secure machine learning [17, 22, 23], Docker container [3], distributed system [11, 25] and and database system [5, 21].

Sensor virtualization has been explored in the domain of IoT networks [4, 8]. Senaas [1] uses event-driven sensor virtualization technology to provide interface for IoT clouds. SenseWrap [7] and Kim el. [14] designs a network middleware to provide cloud-based IoT virtualization. Furthermore, Islam el. [12] designed virtualization approach for wireless sensor networks. Ko el. [15] proposed a virtualization of sensor network management on mobile devices.

Some other work designed sensor or actuator virtualization framework on specific areas, such as smart camera [13, 24]. However, unlike *VirtSense* , none of the above work focuses on the sensor virtualization on single node. PROTC [20] uses ARM TrustZone to protect drone's safety. SeCloak [16] uses ARM TrustZone to ensure an actuation control on mobile device. Unlike the prior work, *VirtSense* uses enclave to achieve a secure virtualization environment for sensors.

7 CONCLUSION

We proposed a new sensing architecture *VirtSense* based on virtualization and enclave to satisfy the needs of multi-tenancy sensing applications. Through our preliminary results, we show that *VirtSense* provides both secure and flexing sensing system.

ACKNOWLEDGEMENT

This research was supported in part by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, and by the National Science Foundation under awards # CNS-1329755 and CNS-1705135. Any findings in this material are those of the author(s) and do not reflect the views of any of the above funding agencies. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

REFERENCES

[1] S. Alam, M. M. R. Chowdhury, and J. Noll. 2010. SenaaS: An event-driven sensor virtualization approach for Internet of Things cloud. In *2010 IEEE International Conference on Networked Embedded Systems for Enterprise Applications*. 1–6. https://doi.org/10.1109/NESEA.2010.5678060

[2] A. Aldroubi and K. GrÃűchenig. 2001. Nonuniform Sampling and Reconstruction in Shift-Invariant Spaces. *SIAM Rev.* 43, 4 (2001), 585–620. https://doi.org/10.1137/S0036144501386986 arXiv:https://doi.org/10.1137/S0036144501386986

[3] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark L. Stillwell, David Goltzsche, Dave Eyers, Rüdiger Kapitza, Peter Pietzuch, and Christof Fetzer. 2016. SCONE: Secure Linux Containers with Intel SGX. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Savannah, GA, 689–703. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/arnautov

[4] Sunanda Bose, Atrayee Gupta, Sriyanjana Adhikary, and Nandini Mukherjee. 2015. Towards a Sensor-Cloud Infrastructure with Sensor Virtualization. In *Proceedings of the Second Workshop on Mobile Sensing, Computing and Communication (MSCC '15)*. ACM, New York, NY, USA, 25–30. https://doi.org/10.1145/2757743.2757748

[5] Helena Brekalo, Raoul Strackx, and Frank Piessens. 2016. Mitigating Password Database Breaches with Intel SGX. In *Proceedings of the 1st Workshop on System Software for Trusted Execution (SysTEX '16)*. ACM, New York, NY, USA, Article 1, 6 pages. https://doi.org/10.1145/3007788.3007789

[6] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. 2010. TaintDroid: An Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*. USENIX Association, Berkeley, CA, USA, 393–407. http://dl.acm.org/citation.cfm?id=1924943.1924971

[7] P. Evensen and H. Meling. 2009. SenseWrap: A service oriented middleware with sensor virtualization and self-configuration. In *2009 International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*. 261–266. https://doi.org/10.1109/ISSNIP.2009.5416827

[8] Pål Evensen and Hein Meling. 2009. Sensor Virtualization with Self-configuration and Flexible Interactions. In *Proceedings of the 3rd ACM International Workshop on Context-Awareness for Self-Managing Systems (Casemans '09)*. ACM, New York, NY, USA, 31–38. https://doi.org/10.1145/1538864.1538870

[9] Karlheinz GrÃűchenig and Harald Schwab. 2003. Fast Local Reconstruction Methods for Nonuniform Sampling in Shift Invariant Spaces. *SIAM J. Matrix Anal. Appl.* 2002 (2003), 24–899.

[10] Frida Gunnarsson, Fredrik Gustafsson, and Fredrik Gunnarsson. 2004. Frequency Analysis Using Non-Uniform Sampling With Application To Active Queue Management. (2004).

[11] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. 2016. Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Savannah, GA, 533–549. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/hunt

[12] Md. Motaharul Islam, Jun Hyuk Lee, and Eui-Nam Huh. 2013. An Efficient Model for Smart Home by the Virtualization of Wireless Sensor Network. *International Journal of Distributed Sensor Networks* 9, 2 (2013), 168735. https://doi.org/10.1155/2013/168735

[13] Shubham Jain, Viet Nguyen, Marco Gruteser, and Paramvir Bahl. 2017. Panoptes: Servicing Multiple Applications Simultaneously Using Steerable Cameras. In *Proceedings of the 16th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN '17)*. ACM, New York, NY, USA, 119–130. https://doi.org/10.1145/3055031.3055085

[14] S. H. Kim and D. Kim. 2015. Enabling Multi-Tenancy via Middleware-Level Virtualization with Organization Management in the Cloud of Things. *IEEE Transactions on Services Computing* 8, 6 (Nov 2015), 971–984. https://doi.org/10.1109/TSC.2014.2355828

[15] JeongGil Ko, Byung-Bog Lee, Kyesun Lee, Sang Gi Hong, Naesoo Kim, and Jeongyeup Paek. 2015. Sensor Virtualization Module: Virtualizing IoT Devices on Mobile Smartphones for Effective Sensor Data Management. *International Journal of Distributed Sensor Networks* 11, 10 (2015), 730762. https://doi.org/10.1155/2015/730762 arXiv:https://doi.org/10.1155/2015/730762

[16] Matthew Lentz, Rijurekha Sen, Peter Druschel, and Bobby Bhattacharjee. 2018. SeCloak: ARM Trustzone-based Mobile Peripheral Control. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '18)*. ACM, New York, NY, USA, 1–13. https://doi.org/10.1145/3210240.3210334

[17] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. 2017. Oblivious Neural Network Predictions via MiniONN Transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. ACM, New York, NY, USA, 619–631. https://doi.org/10.1145/3133956.3134056

[18] Renju Liu, Lintong Jiang, Ningzhe Jiang, and Felix Xiaozhu Lin. 2015. Anatomizing System Activities on Interactive Wearable Devices. In *Proceedings of the 6th Asia-Pacific Workshop on Systems (APSys '15)*. ACM, New York, NY, USA, Article 18, 7 pages. https://doi.org/10.1145/2797022.2797032

[19] Renju Liu and Felix Xiaozhu Lin. 2016. Understanding the Characteristics of Android Wear OS. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '16)*. ACM, New York, NY, USA, 151–164. https://doi.org/10.1145/2906388.2906398

[20] Renju Liu and Mani Srivastava. 2017. PROTC: PROTeCting Drone's Peripherals Through ARM TrustZone. In *Proceedings of the 3rd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications (DroNet '17)*. ACM, New York, NY, USA, 1–6. https://doi.org/10.1145/3086439.3086443

[21] Kai Mast, Lequn Chen, and Emin Gün Sirer. 2017. Scaling Databases Through Trusted Hardware Proxies. In *Proceedings of the 2Nd Workshop on System Software for Trusted Execution (SysTEX'17)*. ACM, New York, NY, USA, Article 9, 6 pages. https://doi.org/10.1145/3152701.3152712

[22] Olga Ohrimenko, Felix Schuster, Cedric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. 2016. Oblivious Multi-Party Machine Learning on Trusted Processors. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX, 619–636. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/ohrimenko

[23] Nicolas Papernot, Patrick D. McDaniel, Arunesh Sinha, and Michael P. Wellman. 2016. Towards the Science of Security and Privacy in Machine Learning. *CoRR* abs/1611.03814 (2016). arXiv:1611.03814 http://arxiv.org/abs/1611.03814

[24] Navin K. Sharma, David E. Irwin, Prashant J. Shenoy, and Michael Zink. 2011. MultiSense: Fine-grained Multiplexing for Steerable Camera Sensor Networks. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems (MMSys '11)*. ACM, New York, NY, USA, 23–34. https://doi.org/10.1145/1943552.1943556

[25] Wenting Zheng, Ankur Dave, Jethro G. Beekman, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. 2017. Opaque: An Oblivious and Encrypted Distributed Analytics Platform. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 283–298. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/zheng