Secure Edge Computing with ARM TrustZone

Robert Pettersen, Håvard D. Johansen and Dag Johansen
University of Tromsø, The Arctic University of Norway, Tromsø, Norway

Keywords: IoT, ARM TrustZone, Intel SGX, Secure Enclave, Trusted Execution, Edge Computing, Cloud Computing.

Abstract: When connecting Internet of Things (IOT) devices and other Internet edge computers to remote back-end hybrid or pure public cloud solutions, providing a high level of security and privacy is critical. With billions of such additional client devices rapidly being deployed and connected, numerous new security vulnerabilities and attack vectors are emerging. This paper address this concern with security as a first-order design principle: how to architect a secure and integrated middleware system spanning from IOT edge devices to back-end cloud servers. We report on our initial experiences from building a prototype utilizing secure enclave technologies on IOT devices. Our initial results indicate that isolating execution on ARM TrustZone processors comes at a relatively negligible cost.

1 INTRODUCTION

Internet application and service components are currently being relocated from central cloud backend services to less resource-rich IOT and mobile devices at the edges of the Internet. Such *edge computing* is a natural evolutionary step in distributed systems architectures, and is at the heart of the "4th Industrial Revolution" currently unfolding (Schwab, 2016). We are about to witness a paradigm shift where computer-science innovations have the potential to disrupt society at a scale never witnessed before by tapping into the wide variety of data generated by edge devices.

Because IOT and mobile devices are designed to operate in physical proximity to end-users, they give hostile entities ample opportunities to tinker with the hardware and software stacks. Such malicious modification to the edge devices can lead to unspecified system behavior and new attack vectors. This inherent weakness of edge computing is particularly concerning for applications that monitor and control IOT devices in modern buildings and critical infrastructure (e.g., power grid), or applications that collect and analyze sensitive personal information from wearable or ambient lifelogging devices, like surveillance cameras or smart watches (Johansen et al., 2015a).

In response to the rising need of modern distributed-system architecture to securely execute software remotely on untrusted hardware, several hardware vendors have devised new on-chip security measures in their products. Most notably are perhaps the Intel Software Guard Extensions (SGX) (Anati et al., 2013) and ARM TrustZone technologies (ARM Limited, 2009; Ngabonziza et al., 2016; Shuja et al., 2016). Although their underlying hardware design, features, and interfaces differ substantially, both SGX and TrustZone essentially provide the same key concepts of hardware isolated execution domains, which we commonly refer to as *enclaves*, and the ability to bootstrap attested software stacks into those enclaves. These features mitigate need to include the Operating System (OS) and any hypervisor in the Trusted Computing Base (TCB): a previous major security problem for both cloud and edge-computing systems. Note also that SGX and TrustZone only targets enforcement of confidentiality and integrity policies, and not availability or liveness. The host OS can still halt execution of protected software, delay calls when they cross execution domains (or into the OS kernel), and ignore incoming network packets.

This paper reports on our initial experiences incorporating ARM TrustZone technology into *Diggi*:[1] a generic IOT/mobile/cloud platform that enables IOT devices, mobile phones, smart-home computers, personal computers, hybrid-cloud solutions, and proprietary cloud solutions from different vendors to seamlessly connect and integrate in a privacy-preserving and secure manner. Application programmers would normally need to decide at design time where data should be cached and stored, and where computations should be executed. However, Diggi attempt to bet-

[1] Diggi translates to the word "thing" in the indigenous arctic North-Sami language.

Pettersen, R., Johansen, H. and Johansen, D.
Secure Edge Computing with ARM TrustZone.
DOI: 10.5220/0006308601020109
In *Proceedings of the 2nd International Conference on Internet of Things, Big Data and Security (IoTBDS 2017)*, pages 102-109
ISBN: 978-989-758-245-5

ter utilize resources at edge-nodes by making those decisions just-in-time by shipping functional *meta-code* components between computational devices at run-time. Understanding how Diggi can best utilize existing hardware isolation mechanisms, like ARM TrustZones, for secure remote execution of meta-code components, is essential for Diggi and therefore the main topic of this paper.

The rest of the paper is organized as follows. Section 2 outlines the broader architecture and motivation for using secure enclave technologies as a fundamental building block in Diggi. Section 3 details Diggi core design related to ARM TrustZone. Section 4 reports on initial, general performance results, while Section 5 presents related work. Finally, Section 6 concludes.

2 Diggi OVERVIEW

Diggi is a middleware system for secure federation of IOT and other edge devices with backend enterprise and public cloud services. Central concepts to enable this includes meta-code and secure enclave technologies.

2.1 Architecture

We are developing Diggi towards an open-source infrastructure seamlessly integrating IOT/mobile/cloud systems utilizing secure enclave technologies. Code and data within an enclave are protected by the hardware from disclosure or modification from any thread executing on the outside: providing strict confidentiality and integrity guarantees. Both ARM TrustZone and Intel SGX enable such protections through the use of on-chip features that sets up isolated execution domains into which the application programmer can load his code and data. Execution threads external to an enclave cannot read or tamper with that enclave's execution or data. Data may be taken out of an enclave and stored externally, but typically only in encrypted form.

We are currently using Diggi as a middleware glue between secure enclaves residing in this very heterogeneous network of IOT devices, personal computers, enterprise clouds, and public clouds. ARM TrustZone is developed for lightly equipped devices, while Intel SGX can span the broader spectrum of more resource rich computers. Hence, an integration of and combination of both these state-of art secure technologies from ARM and Intel ensure that the same security-sensitive applications can be deployed in a very heterogeneous distributed system with the same security property guarantees. Secure communication channels are established between the different Diggi nodes.

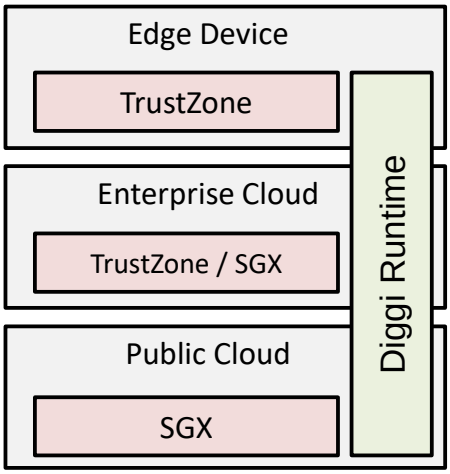

Figure 1: Secure integration of edge devices with enterprise and public clouds using trusted hardware.

The Diggi architecture consists of three vertically stacked layers, as shown in Figure 1. The top-most layer contains the edge client devices and is primarily based on ARM TrustZone. This can be, for instance, an IOT sensor device or a cellular phone. Intel SGX already runs in, for instance, notebooks and can be a client device as well. The middle layer is a client federating layer typically in the same administrative domain as the client devices it connects to. As such, this can be a private SGX server or an enterprise cloud. By connecting to the third layer containing SGX nodes, the public clouds, a hybrid cloud solution with one cloud provider can be configured. Alternatively, a multi-cloud deployment is possible by connecting to two different public cloud providers. We have previously built Balava (Nordal et al., 2011), a three-layer multi-cloud storage system based on meta-code spanning such a deployment, but this was prior to the emergence of Intel SGX and ARM TrustZone. However, Diggi also builds on the concept of meta-code, which we will discuss next.

2.2 Meta-code

The meta-code concept (Hurley and Johansen, 2014) is a convenient structuring mechanism for supporting autonomic and distributed computing in this broad domain. The idea is to attach fine level, autonomic meta-code processes to any data element being accessed. The meta-code is distributed, installed, and revoked as mobile-code components that execute either periodically or on data access. This way, user-

Table 1: Example meta-code policies applicable to common data operations.

Operation	Meta-code
open()	Download an up-to-date remote access-control list. Decrypt object into enclave memory only if authorized by access-control list.
read()	De-identify or remove any personal identifiers contained in the returned data. Log data access patterns in a local ledger.
flush()	Send recorded data access patterns from a local ledger to a remote audit server.
write()	Replicate encrypted data to multiple remote cloud hosted sites.

centric functionality can be installed and executed on-demand when, for instance, data from a remote IOT sensor is being read. This can implement upstream evaluation and filtering physically close to a sensor (Carzaniga et al., 2001), highly personalized access control, auditing, and privacy policies (Johansen et al., 2015b; Brasser et al., 2016), or personalized storage systems spanning IOT/mobile/clouds (Nordal et al., 2011). Examples of meta-code for various object operations can be found in Table 1.

Meta-code gives users the powerful ability to customize policies for their data, and enforces those policies without needing to trust applications or their programmers. For example, when attached to a file, the following meta-code grants access to that file for only 24 hours after the file was initially created:

```
if (time() - lstat(path).st_ctime  > 86400){
  throw OSError(EACCES);
}
```

A compelling and promising applicability of meta-code in the IOT/mobile/cloud domain is for the problem of software upgrading. Hundreds of IOT devices might soon be the rule in a modern smart home, and proper safeguards must be put in place to prevent updating IOT interfaces from becoming security holes themselves. Our conjecture is that we can take advantage of secure enclave technologies running on special hardware to remedy the situation. Potentially this protects an application and its secrets from a full compromise of a remote system as well as from a malicious insider. Defined secure regions of code and data will maintain confidentiality even when an attacker has physical control of the node and can conduct direct attacks on memory. The Diggi run-time is designed to allow meta-code and sensitive data to be hosted and run remotely within memory protected from application running on the same machine, including the operating system itself.

We are currently building Diggi for two secure enclave platforms to be integrated into one. One version is for resource-rich cloud servers (Gjerdrum et al., 2016) based on the Intel SGX processor. The other version is for IOT devices based on the ARM TrustZone.

2.3 ARM TrustZone Hardware Layer

The ARM TrustZone technology is a hardware security architecture that can be incorporated into ARMv7-A, ARMv8-A and ARMv8-M System on a Chips (SOCs) (ARM Limited, 2009; Ngabonziza et al., 2016; Shuja et al., 2016). Its main purpose is to enable devices to counter many of the threats that have been difficult to address earlier, such as attacks from the OS running on the device. TrustZone technology provides the infrastructural foundation for the SOC designer to compose a system, choosing between a range of available component, to fulfill specific security requirements. The primary security objective of the architecture is enabling the construction of a programmable environment that allows the confidentiality and integrity of almost any asset to be protected from specific attacks. A platform with these characteristics can be used to build a wide range of cost-effective security solutions, compared to traditional methods where SOC designers utilized proprietary methods.

The TrustZone technology defines two distinct and isolated execution contexts, partitioning all the SOC hardware and software resources into two worlds: the *Normal world*[2] and the *Secure world*. The Normal world hosts the traditional non-trusted OS with its associated processes and subsystems, while the Secure world run trusted software components that enforce confidentiality and integrity constraints in the normal world.

The ARM AMBA3 AXI bus is the primary hardware component enforcing the isolation policy of a TrustZone enabled SOC. This bus matrix extends the 32 bit control signal for each read and write channel with an extra Non-Secure (NS) bit dictating what world each transaction belongs to. By ensuring that all transactions from bus masters in the Normal world have their NS bit set high, insecure information flow from a slave in the secure world can be prevented. In addition to the AMBA3 AXI bus, each physical processor core can securely multiplex instructions from concurrent processes of both world, exposed to the system as two distinct virtual cores: one normal and one secure. This design saves both silicon area and power compared to having a separate core for each

[2]The Normal world is sometimes refereed to as the Non-Secure (NS) world.

world. In a multiprocessor system, cores may operate and transition between worlds independently of one another using hardware interrupts. The mechanism to context switch between processes running in different world is known as *monitor mode*. The virtual core's identity dictates the NS bit used for bus operations, set in the Secure Configuration Register (SCR), and thus also dictates which bus slaves can be accessed. Also, the hardware provides two virtual Memory Management Units (MMUs) extended with NS bits to partition memory between the two worlds. The security of other system components, like tightly coupled memories and the accelerator coherency port, also depend on NS bit to control information flow.

In addition to the main AXI bus, ARM systems may include a lower APB bus, which most peripherals will use. To ensure compatibility with existing devices, the APB bus does not carry the NS bit. It is the AXI-to-APB Bridge's responsibility to manage the NS bit translation to and from connected devices using a special per device signal located on the bus.

The TrustZone framework includes secure boot facilities from a trusted boot loader located in the on-SOC Read-Only Memory (ROM). This ROM is the only system component that cannot be easily changed by an attacker, and is responsible for securely bootstrapping remaining software component into the on-SOC memory.

2.4 Trusted Execution Environments

Because the TrustZone technology only distinguish two protection domains at the hardware level, concurrent mutually distrusting applications must be further isolated in software. Such Trusted Execution Evironments (TEEs) can range from simple libraries, to a full operating system kernel. In the latter case, the TrustZone system will be running two concurrent operating-system kernels: one in the Normal world and one in the Secure world. By using the MMU's ability to partition Secure world memory, the Secure world kernel can run concurrent and mutually distrusting application in separate user-space sandboxes.

While the ARM TrustZone architecture specifies mechanisms to handle secure interrupts, memory and peripherals, it is up to the SOC designer to implement these. Vendors can choose to implement all the mechanisms, or only a subset of them, to provide the security mechanisms needed for devices that will utilize the specific SOC. In our case, Diggi aims to follow the GlobalPlatform Trusted Execution Evironment (TEE) client API specification 1.0 (Global Platform, 2011).

3 DESIGN AND IMPLEMENTATION

Diggi aims to utilize secure enclave technology to provide secure communication, provisioning and execution between the cloud, mobile, and IOT devices. We start by implementing the Diggi architecture running on IOT devices with TrustZone technology. The Raspberry Pi 3 model b[3] embedded development board was chosen as a starting point for our implementation. The Raspberry Pi has a SOC which only provides TrustZone exception states, the mechanisms and hardware required to implement secure memory and peripherals are not available.

TrustZone exception states enable the Raspberry Pi to execute code both in the Normal and Secure worlds. Allocating memory, and accessing storage and network interfaces cannot be performed in the Secure world and need to be emulated. Requests for memory and access to peripherals are routed to a process running in the Normal world called the *TEE Supplicant*. The supplicant process will perform the request in the Normal world on behalf of the Secure world, and will not offer the security and isolation that the TrustZone architecture specifies.

The Raspberry Pi runs Linaro's[4] OP-TEE OS[5] in the Secure world, and a 64 bit Linux kernel version 4 running in the Normal world with a TEE driver to facilitate communication between worlds. The Secure world OS is accessible from the Normal world OS using the GlobalPlatform TEE client API specification 1.0 (Global Platform, 2011), which is also used to trigger execution of applications within the Secure world.

Our previous work with dynamic resolution of code (Valvåg et al., 2016) was used to create mechanisms for dynamically resolving trusted application dependencies in the Secure world OS. Each trusted application is identified by a 128 bit Universally Unique Identifier (UUID), which is used in the run-time dependency resolution process. The process starts by installing an application on the device through the normal appstore associated with the OS running in the Normal world (se Figure 2). The appstore can be Google Play, Apple's iTunes store, the Ubuntu's apt-get, or a similar well established channel for distribution of native applications.

When the application needs to execute in the Secure world, a request is sent through the TEE driver to the Secure world OS. The request will contain the

[3] http://www.raspberrypi.org

[4] http://www.linaro.org

[5] https://github.com/OP-TEE/optee_os

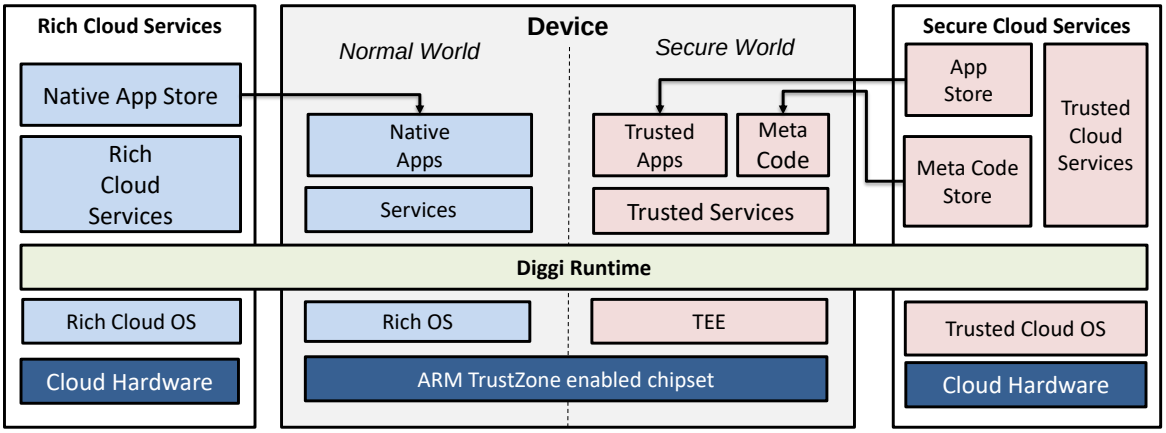

Figure 2: Outline of application provisioning for IOT devices in Diggi. The Normal world part of the application is distributed from a cloud-hosted appstore to the Normal world OS. When the application requests trusted execution or access meta-code protected data, the Secure world will request the secure part of the application through the secure appstore and do validation of the code before installing it and allowing requests between worlds.

UUID of the trusted application. If the trusted application is not installed on the device, which will be the case at the first invocation, the Secure world OS will utilize Diggi resolution service to locate the trusted application. Diggi will establish a connection to a secure appstore and retrieve the application associated with the UUID. The application will be validated before it is installed and loaded in the Secure world. After the application has been successfully installed, the request from the native application will be delivered for processing. Figure 2 illustrates this architecture.

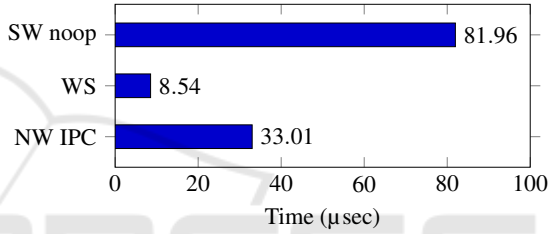

Figure 3: Mean end-to-end execution times for a noop Secure world invocation executed 1000 times, compared to traditional IPC between Normal world processes through named pipes.

4 PERFORMANCE EVALUATION

For our performance evaluation, we chose the Raspberry Pi 3 model b embedded development board to host Diggi. The Raspberry Pi has 1 GB RAM and a 1.2 GHz quad-core ARM Corex-A53 CPU and provides TrustZone exception states which enable execution in the Secure world. The mechanisms and hardware required to implement secure memory and peripherals are not available. Our experiments in this paper are therefore focused on transitions between the Normal and the Secure world, and execution in the Secure world.

4.1 World-switch Overhead

In our first experiment, we measure the end-to-end overhead of switching from executing in the Normal world to executing in the Secure world inside a TEE hosted application. Such world switches are necessary to achieve secure execution of code, and knowledge about their performance characteristics are important when deciding how an application or system is best partitioned between the worlds.

Conceptually, communication between applications in the Normal world and the Secure world can be compared to Interprocess Communication (IPC) between two processes. We measure the end-to-end delay for IPC communication between two processes in the Normal world to get a baseline communication cost between processes.

All measurements have been executed 1000 times, and we present the mean. Figure 3 shows the measured mean execution time for our noop Secure world invocation. The observed mean end-to-end time is 81.96 µs. Included in this is two world changes, one from the Normal world to the Secure world, and one from the Secure world to the Normal world. The observed mean time for a single world change is 8.54 µs. For comparison, the mean end-to-end time for IPC between two Normal world processes is 33.01 µs.

The time to switch worlds only amounts to 17.08 µs of the total time for the noop invocation. By instrumenting the code path from the start of the invocation to the end, a breakdown of where time

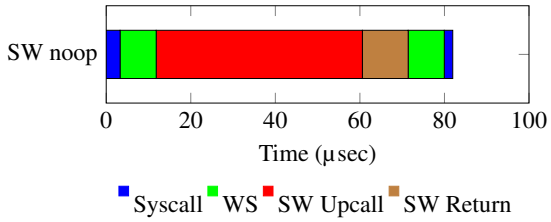

Figure 4: Breakdown of the Secure world noop invocation.

was spent was produced, as seen in Figure 4. Here, 48.77 µs of the total time is spent in the Secure world OS and switching to the Secure world user level.

4.2 Execution Overhead

The second experiment compares execution times in the Secure world to the Normal world. Since the Secure world is expected to perform cryptographic operations we chose to execute a series of cryptographic hashing operations to evaluate the execution performance of the Secure world.

Linaro's OP-TEE OS has adapted LibTomCrypt[6] to provide cryptographic operations through the GlobalPlatform Application Programming Interfaces (APIs) for applications in the Secure world. The same library was used when executing the cryptographic operations in the Normal world.

SHA-256 was used to hash a variable sized buffer 1000 times, both in the Secure world and the Normal world. We have omitted the time needed for world switches and copying of data, since the Raspberry Pi does not support secure memory. The mean execution times is presented in Figure 5. The results show no negative impact of hashing in the Secure world.

The ARM TrustZone architecture includes efficient hardware cryptographic engines that can accelerate security and cryptographic operations. Unfortunately, the Raspberry Pi does not include this hardware and the effects from this hardware cannot be seen in the graphs.

5 RELATED WORK

Mechanisms for providing a trusted execution environment is not new. Popular banking services have already utilized the Secure Element (SE) in SIM cards to provide secure authentication for users of their bank services. The Apple A7 (and later) SOC utilizes separate ARM processor cores to provide a trusted execution environment to store and authenticate user fingerprints both for the device and supported applications.

[6]https://github.com/libtom/libtomcrypt

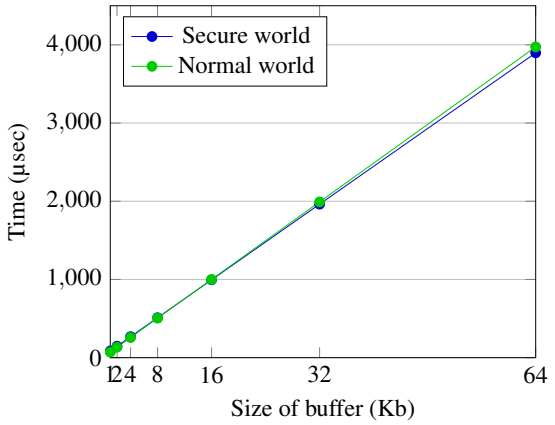

Figure 5: Mean execution times for SHA-256 hashing a variable sized buffer in the Secure world, compared to hashing in the Normal world.

Smartcards have frequently been deployed in systems that need a high-degree of security assurance. Such cards, however, have limited processing, bandwidth, and memory capacity; and cannot directly interact with the user to, for instance, obtain personal identification numbers. As such, smartcards depend on untrusted OS logic for some of their operations. Trusted Platform Modules (TPMs) (TCG Published, 2011) are chips typically permanently embedded on a system's mainboard, providing similar features as smartcards. TPMs were initially specified by the Trusted Computing Group (TCG), and now approved as the standardized specification ISO/IEC 11889.

In the Trusted Language Runtime (TLR) (Santos et al., 2014), programmers can specify which of the classes of a .NET program, or trustlets, that should run within the secure world, and how they are to interface with the normal world. Although the primitives described in this work is highly relevant for Diggi, it is unclear how the proposed trustlets mechanism, and their trustboxes execution environments, can support our more dynamic and mobile meta-code concept.

The PrivateZone framework (Jang et al., 2016) address the limited ability of the TrustZone hardware to host mutually distrusting applications within a single TEE OS. For this, PrivateZone sets up private execution environments in the normal word using the secure world monitoring mode, controlled by a normal world kernel driver, to switch between the different execution contexts. The approach requires some modification to the underlying normal world OS that may prevent practical deployment.

Conceptually, sandboxing (Goldberg et al., 1996) relates to our work by providing a security mechanism for separating running programs. Diggi separates software execution in a trusted Secure world and an untrusted Normal world. Similar structuring

can be done using sandboxes with code running inside or outside a sandbox. A sandbox, however, controls the resources an application can use securing the local computer; a secure enclave though secures the application from security violations originating from software at the computer.

TaintDroid (Enck et al., 2014) extends Google's Android kernel to track third-party binary applications dynamically at run-time by labeling data from possible sensitive sources as tainted during execution. The taint is encoded using a single bit on certain memory regions, similarly how the TrustZone hardware use the NS-bit on memory and buses to create two execution domains. Unlike the TrustZone technology, TaintDroid leverages facilities in the Java virtual machine instead of the hardware level to label data as either tainted or not. The taint follows the data as it propagates through program variables, inter-process communication, and file IO, and enables TaintDroid to monitor and control that sensitive data is not transmitted out of the phone. Because taint is encoded as a single bit on certain memory regions, isolating multiple execution domains would require additional mechanisms, as with ARM TrustZone. Automated partitioning of Android applications to run on TrustZone hardware has also been suggested (Rubinov et al., 2016).

In the security-typed Java derived programming language JiF (Sabelfeld and Myers, 2003), programmers can express rich information-flow control policies as labels attached to variables at the source code level to express restriction on information flow within the program. This include setting up multiple execution privilege levels and policies on how data can flow between those. The compiler analyzes the code to make sure that information flows in accordance with those policies, and admits only complying programs. Although expressive, these language based methods cannot secure legacy applications and require the compiler to be part of the TCB. Some kernels, like Asbestos (Efstathopoulos et al., 2005), use labels expressed in terms of tags combined with capabilities to enforce information-flow security policies between hosted, but are currently only available in prototype form.

Secure Data Capsules (Maniatis et al., 2011) are data objects associated with a policy tag that defines the provenance and a usage policy, similarly to our meta code. These policy tags are cryptographically bound to the associated data, and derived data are tagged with a derived policy tag. Untrusted applications can operate on tagged objects only while inside a secure execution environment that enforces and propagates policy tags.

6 CONCLUSION

We cannot naively assume that computers running IOT software are totally trustworthy. Despite use of state-of-the-art authentication and access control mechanisms from distributed systems, we might need to minimize what components to trust and even regard the OS and co-located applications on a single IOT device with suspicion and care.

Integrating IOT devices with other client computers or remote cloud servers opens the overall system for unwanted security attack vectors that should be avoided. This might include that IOT devices have USB ports probably not needed for operation of the device, devices are out of sight for users and might be physically tampered with, and functionality not required for the current deployment is available via an API layer. The list is, unfortunately, very long and growing.

We are investigating how to properly secure such an IOT infrastructure and have adopted a rigorous security-by-design strategy. Our current IOT prototype is therefore built around secure hardware from scratch with a trusted module known as secure enclave technology at its foundation. We use Intel SGX for the resource-rich cloud servers, and we use ARM TrustZone for IOT devices to enforce security isolation on even a layer below the operating system. This reduces the trusted computing platform to a minimum; to the vendors Intel and ARM that in any case must be trusted if you want to algorithmically automate any task of interest.

Securing a heterogeneous distributed system with IOT devices at the edges and cloud servers at the core is a daunting task. Our humble attitude includes accepting that secure hardware is just a piece in the holistic security puzzle. It is fundamentally important though, and will gradually be complemented with our on-going work on utilizing secure and encrypted storage, in our work on making upgrades to IOT devices secure during the lifetime of the device through meta-code, and how to protect data integrity while in transit to and from the IOT devices. Each such approach builds greater security assurance in the overall infrastructure, and we are confident that secure enclave technologies like reported in this paper will play a pivotal role.

ACKNOWLEDGMENTS

This work was supported in part by the Norwegian Research Council project numbers 231687/F20, and the Corpore Sano Centre at UiT, Norway. We would

like to thank the anonymous reviewers for their useful insights and comments.

REFERENCES

Anati, I., Gueron, S., Johnson, S., and Scarlata, V. (2013). Innovative technology for cpu based attestation and sealing. In *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, volume 13.

ARM Limited (2009). ARM security technology: Building a secure system using TrustZone technology. White paper PRD29-GENC-009492C, ARM Limited.

Brasser, F., Kim, D., Liebchen, C., Ganapathy, V., Iftode, L., and Sadeghi, A.-R. (2016). Regulating arm trustzone devices in restricted spaces. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 413–425. ACM.

Carzaniga, A., Rosenblum, D. S., and Wolf, A. L. (2001). Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems (TOCS)*, 19(3):332–383.

Efstathopoulos, P., Krohn, M., VanDeBogart, S., Frey, C., Ziegler, D., Kohler, E., Mazières, D., Kaashoek, F., and Morris, R. (2005). Labels and event processes in the asbestos operating system. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles*, SOSP '05, pages 17–30, New York, NY, USA. ACM.

Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B.-G., Cox, L. P., Jung, J., McDaniel, P., and Sheth, A. N. (2014). Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, 32(2):5.

Gjerdrum, A. T., Håvard, D., and Johansen, D. (2016). Implementing informed consent as information-flow policies for secure analytics on ehealth data: Principles and practices. In *Connected Health: Applications, Systems and Engineering Technologies (CHASE), 2016 IEEE First International Conference on*, pages 107–112. IEEE.

Global Platform (2011). TEE system architecture. *Global Platform technical overview*.

Goldberg, I., Wagner, D., Thomas, R., Brewer, E. A., et al. (1996). A secure environment for untrusted helper applications: Confining the wily hacker. In *Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography*, volume 6, pages 1–1.

Hurley, J. and Johansen, D. (2014). Self-managing data in the clouds. In *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, pages 417–423. IEEE.

Jang, J., Choi, C., Lee, J., Kwak, N., Lee, S., Choi, Y., and Kang, B. (2016). Privatezone: Providing a private execution environment using arm trustzone. *IEEE Transactions on Dependable and Secure Computing*.

Johansen, H., Gurrin, C., and Johansen, D. (2015a). Towards consent-based lifelogging in sport analytic. In *MMM 2015, Part II*, number 8936, pages 335–344. Springer International Publishing.

Johansen, H. D., Birrell, E., Van Renesse, R., Schneider, F. B., Stenhaug, M., and Johansen, D. (2015b). Enforcing privacy policies with meta-code. In *Proceedings of the 6th Asia-Pacific Workshop on Systems*, page 16. ACM.

Maniatis, P., Akhawe, D., Fall, K., Shi, E., McCamant, S., and Song, D. (2011). Do you know where your data are? Secure data capsules for deployable data protection. In *of the 13th USENIX Conference on Hot Topics in Operating Systems*, HotOS '11, pages 22–27. USENIX Association.

Ngabonziza, B., Martin, D., Bailey, A., Cho, H., and Martin, S. (2016). Trustzone explained: Architectural features and use cases. In *Collaboration and Internet Computing (CIC), 2016 IEEE 2nd International Conference on*, pages 445–451. IEEE.

Nordal, A., Kvalnes, Å., Hurley, J., and Johansen, D. (2011). Balava: Federating private and public clouds. In *Services (SERVICES), 2011 IEEE World Congress on*, pages 569 –577.

Rubinov, K., Rosculete, L., Mitra, T., and Roychoudhury, A. (2016). Automated partitioning of android applications for trusted execution environments. In *Proceedings of the 38th International Conference on Software Engineering*, pages 923–934. ACM.

Sabelfeld, A. and Myers, A. (2003). Language-based information-flow security. *Selected Areas in Communications, IEEE Journal on*, 21(1):5–19.

Santos, N., Raj, H., Saroiu, S., and Wolman, A. (2014). Using arm trustzone to build a trusted language runtime for mobile applications. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 67–80, New York, NY, USA. ACM.

Schwab, K. (2016). The fourth industrial revolution. World Economic Forum Geneva.

Shuja, J., Gani, A., Bilal, K., Khan, A. U. R., Madani, S. A., Khan, S. U., and Zomaya, A. Y. (2016). A survey of mobile device virtualization: taxonomy and state of the art. *ACM Computing Surveys (CSUR)*, 49(1):1.

TCG Published (2011). TPM main part 1 design principles. Specification Version 1.2 Revision 116, Trusted Computing Group,.

Valvåg, S. V., Pettersen, R., Johansen, H., and Johansen, D. (2016). Lady: Dynamic resolution of assemblies for extensible and distributed .net applications. In *CLOSER 2016 : Proceedings of the 6th International Conference on Cloud Computing and Services Science*, pages 118–128.