Canonical

An introduction to confidential computing

Learn how to improve the security of your public cloud workloads with this privacy-enhancing technology

Executive summary

Developers can spend a lot of time tightening up the security of an application through static code analysis, penetration testing, multi-factor authentication, and more. No matter how secure their user-level application is, however, those security guarantees are only one piece of the puzzle once it's running on a platform in production. At that point, threats can still come from the millions of lines of code within the platform's privileged system software, such as the operating system, virtual machine manager, or firmware. To function, privileged system software has unrestricted access to all the resources of unprivileged user-level applications, but this makes it a valuable target for exploits.

Thankfully, there exists today a solution to ensure the privacy of workloads in the cloud: trusted execution environments (TEE), more commonly known today as confidential computing. This new primitive is here to give you back control over the security guarantees of your code and data. It achieves this by allowing you to run your workload within a logically isolated hardware-rooted execution environment. As such, your workload's code and data are better protected against the cloud's administrator and privileged system software.

At Canonical, we believe that confidential computing and privacy enhancing technologies will be the default way of doing computing in the future. This is why our confidential computing portfolio is free on all public clouds.

This paper will help you better understand the challenge of run-time security, and explain how popular TEEs implement isolation and remote attestation. It will also provide you with an overview of Ubuntu's confidential computing portfolio. This will be especially useful if you are using Ubuntu or considering it for your next project.

The security of public cloud deployments

Cloud technology and virtualisation enable developers and administrators to deploy infrastructure at a pace previously unheard of, which has brought them huge gains and enabled almost anyone to create internet services. However, this facility and flexibility does not mask the underlying need for deployments to be secure. In fact, the security of your workload may very well be compromised by external malicious actors who could exploit vulnerabilities within the cloud infrastructure, as well as potentially rogue internal cloud administrators with privileged access to your machine or virtual environment.

Therefore, to keep their public cloud workloads secure, users need to have visibility over the lifecycle of their data and protect it at all its stages. At a high-level, data can be in one of the following states, as illustrated in Figure 1:

1. In-transit: For sending data to the public cloud over insecure networks, customers can use secure protocols such as TLS.

2. At-rest: To securely store the data when it is at rest and sitting idle in the public cloud's storage, customers can encrypt their data with a key that is generated and managed by them, and further protect it using the cloud's hardware security modules.

3. In-use: When it is time to compute over the data, the public cloud provider needs to first decrypt it and then move it in cleartext from the server's secondary storage and into its system memory, DRAM.

In transit	At rest	In use
TLS	Disk encryption	?

Figure 1: Trust and the public cloud

Computing over the data is unavoidable. After all, this is why customers use the public cloud: to take advantage of its elasticity and great computational resources. Once in system memory, however, customers' code and data can be compromised by a vulnerable or malicious system level software (OS, hypervisor, Bios) as illustrated in figure 2, or even by a malicious cloud operator staff with administrator or physical access to their platforms. Indeed, the security of user-level applications depends on the security of its underlying system software. This is because privileged system software gets unrestricted access to all the resources of unprivileged user-level applications as it controls its execution, memory, and access to the underlying hardware. It's a feature, not a bug!

Ring 3	Userland
Ring 2	“Drivers”
Ring 1	Guest OS
Ring 0	Kernel
Ring -1	Hypervisor
Ring -2	System Management Mode

Figure 2: The hierarchy of the computing software stack

The security of public cloud deployments

As we have laid out above, run-time security is a tough problem. You want the cloud to analyse your workloads without learning anything about the content of that very particular data. And you want the cloud’s privileged system software to manage the lifecycle of your workload, but have no impact on its security guarantees. How can you compute over data without actually looking at its values? And how can you expect a vulnerable hypervisor not to threaten the security of the user-level applications it runs?

This challenge of resolving the tension between confidentiality and utility has been actively researched for many years, and is known as privacy enhancing technologies, PETs. PETs can be defined as the range of technologies that help us resolve the tension between data privacy and utility. They achieve this by allowing us to compute on data and derive value from it, while also preserving its privacy. This is unlike traditional cryptographic primitives, such as AES (advanced encryption standard), which only allow us to preserve data confidentiality, but make it impossible to perform any type of operation on the encrypted ciphertext. PETs can be realised through cryptographic approaches such as differential privacy, homomorphic encryption, secure multiparty computation and zero-knowledge proofs, as well as system approaches like trusted execution environments, otherwise referred to as confidential computing (CC).

Trusted execution environments

The hierarchical architecture of commodity platforms and the insecurity of their privileged system software, make it evident that such an execution environment is unfit for running security-sensitive applications. We name it the general-purpose or rich execution environment, REE[1]. A natural solution to this problem would be to design and deploy more secure commodity system software. While there are some ongoing projects working towards this goal, such as Qubes OS[2] and Coreboot[3], the reality is that currently deployed system software will not be displaced in the foreseeable future, due to two main reasons.

First, there exists a large number of already deployed devices which have strong backward compatibility requirements, and have billions invested in their ecosystems[4] . Second, it is simply difficult to design and implement secure system software, in the face of a constantly changing threat environment, which has become clearly evident during the recent meltdown/spectre vulnerability disclosures[5].

An alternative solution is to run the security-sensitive subset of an application within a separate trusted execution environment (or TEE) that is isolated from the rich execution environment where the untrusted commodity system software runs. In order to be able to reason about TEEs and confidential computing, there are two main primitives that we need to understand: Isolation and remote attestation.

Isolation

Generally speaking, isolation between the TEE and the rest of the platform can be enforced either in software or hardware. However, software solutions such as the ones based on hypervisors are undesirable, because they include untrusted system software in their Trusted Computing Base, TCB. Therefore, confidential computing focuses on hardware-based trusted execution environments, because they provide stronger security guarantees: they can resist software-based attacks and even be resilient, to a certain degree, against physical tampering[6].

The idea of relying on hardware isolation to create a TEE with better security guarantees is not new. Over the years, different ways to realise hardware TEEs have been developed. At a high level, they can be categorised into either physical or logical isolation approaches.

Physical isolation

The code runs within a physically isolated processor, which does not share any context with the untrusted execution environment, the main example of which are secure elements (SEs). These are special-purpose implementations of the traditional smart card. They are designed around a dedicated embedded micro-controller, a set of registers and system memory. Therefore, they can securely execute general-purpose applications, commonly referred to as applets, without interference from the host platform's untrusted system software. SEs can be implemented in two ways as illustrated in figure 3[7]:

1. A discrete hardware module outside the platform's main core. Such SEs are removable and can take two form factors: a Universal Integrated Circuit Card (UICC), commonly known as a SIM card, or a microSD card.

2. An embedded hardware module attached to the platform's motherboard or integrated with its NFC chip, and sharing some of the main system resources such as the memory buses. We refer to such implementations as Embedded Secure Elements, eSE[8].

Such SE-based solutions provide high protection guarantees against the host platform side channel attacks, by virtue of their complete isolation. However, they lack direct access to the system's memory. They are also very constrained in their computational resources.

Multiplexed logical isolation

The security sensitive workloads run within the same host commodity processor, and share its same physical execution context. This is possible because the platform's main processor core can implement several virtual cores which are logically isolated from each other, and can securely multiplex their execution within the same host[9].

Intel Security Guard Extensions (SGX), which was an early realisation of this on X86 platforms, allows ring- 3 applications to control their security by instantiating hardware TEEs, referred to as enclaves, within their own process address space[10]. Enclaves can then securely run the application's security-sensitive code and data, because they isolate it from all other untrusted software running within the platform. Other realisations are Intel SGX, Intel TDX, and AMD SEV on the X86 architecture; TrustZone and the upcoming ARM CCA for the ARM ecosystem; and Keystone for RISC-V architectures, and Nvidia H100 for GPUs.

Multiplexed logical isolation

The security sensitive workloads run within the same host commodity processor, and share its same physical execution context. This is possible because the platform's main processor core can implement several virtual cores which are logically isolated from each other, and can securely multiplex their execution within the same host. Intel Security Guard Extensions (SGX), which was an early realisation of this on X86 platforms, allows ring- 3 applications to control their security by instantiating hardware TEEs, referred to as enclaves, within their own process address space. Enclaves can then securely run the application's security-sensitive code and data, because they isolate it from all other untrusted software running within the platform. Other realisations are Intel SGX, Intel TDX, and AMD SEV on the X86 architecture; TrustZone and the upcoming ARM CCA for the ARM ecosystem; and Keystone for RISC-V architectures, and Nvidia H100 for GPUs.

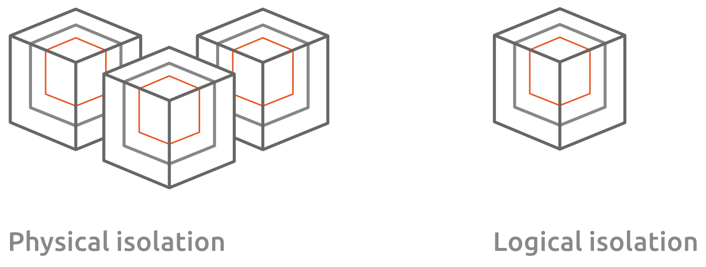

Figure 3: Isolation paradigms

Regardless of their silicon provider, the TEE execution is logically isolated from the main CPU as follows:

1. **Memory isolation through main memory encryption**: instead of bringing the workload's code and data in cleartext to system memory at run-time, many confidential computing-capable CPUs come with a new AES-128 hardware encryption engine embedded within their memory controller which is responsible for encrypting/decrypting memory pages upon every memory read/writE, as illustrated in figure 4. As such, a malicious system administrator who is scraping data from memory, or a vulnerable operating system, can only get access to the encrypted ciphertext. The encryption key is further protected and managed at the hardware level and cannot be accessed by any of the cloud's privileged system software nor its administrators.

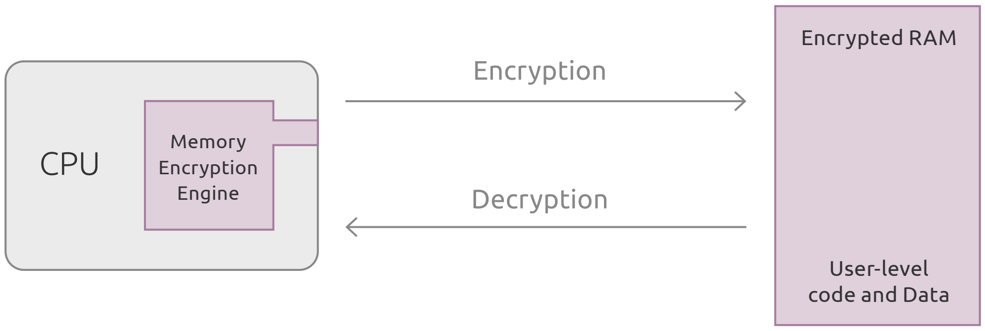

Figure 4: Memory isolation through encryption

2. **Additional CPU-based hardware access control mechanisms**: while encryption protects the confidentiality of the memory pages of confidential workloads, other types of attacks might still be possible. For instance, a malicious host operating system might allocate the same memory page to two different processes. It might also change your encrypted memory values as part of a replay attack, thus breaking the integrity guarantees of your confidential workload. To remedy this, confidential computing-capable CPUs implement new instructions and new data structures that help in auditing the security-sensitive tasks traditionally carried out by the privileged system software, such as memory management and access to the platform's devices. For instance, the new instructions for reading the memory pages mapped to confidential workloads should also return the previous value that was last written into the page in order to mitigate data corruption and replay attacks.

The result of the above described isolation mechanisms, is a TEE enclave where your workload can run securely. For Intel SGX, you need to refactor your application first before you can run its security-sensitive portions of your application in ring-3 as illustrated in Figure 5. For AMD SEV and Intel TDX, your workload will be allowed to run unmodified with a Virtual machine.

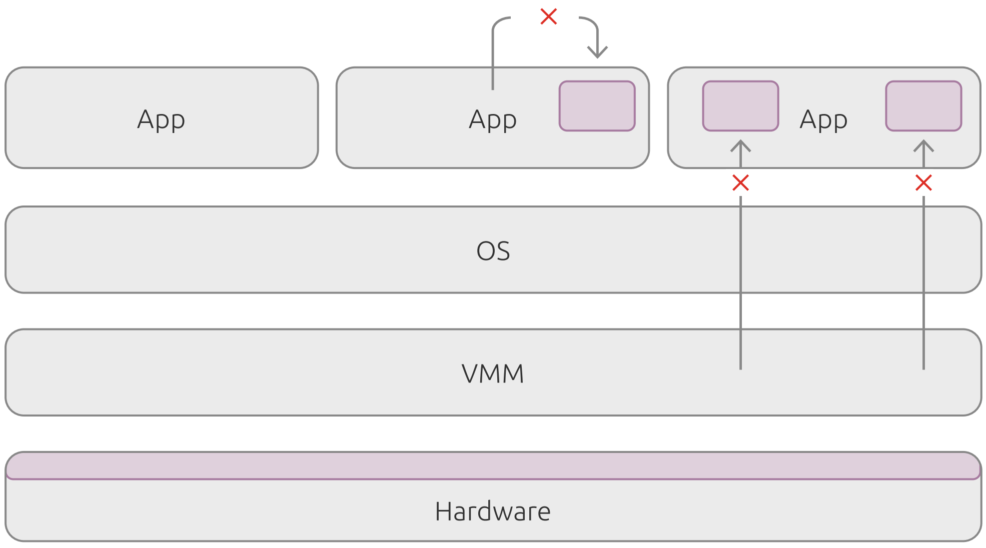

Figure 5: Intel Software Guard Extensionsv

Remote attestation

It's important that the public cloud provider leverages the isolation primitives discussed above in order to run your workload securely within its own isolated trusted execution environment. However, this is not enough. How can you verify that it is not deployed instead in a normal non-confidential way? How can you know that your workload has indeed been provisioned into a genuine hardware TEE? And if so, how can you verify that the cloud's system software has loaded your application as you intended it to be to the TEE? Do you just take the cloud provider's word for it? You don't have to. Instead, you should leverage the remote attestation capabilities of your hardware TEE, before provisioning your secrets into it, and before accepting its results as trustworthy. At a minimum, remote attestation should provide you with a cryptographic proof that consists of:

1. A measurement/hash that attests to the integrity of the software loaded into the TEE.

2. A cryptographic signature over the hash, which attests to the fact that the cloud's TEE hardware used is genuine, and non-revoked.

The remote attestation implementation details depend on both the underlying hardware TEE. For Intel SGX for example, each SGX hardware has a unique attestation key, which is only accessible by special Intel SGX architectural enclaves, such as the quoting enclave. During an attestation protocol, the quoting enclave first forms a structure referred to as a quote, composed of a hash of measurements reflecting the order and the content of code which has been initially loaded into the enclave by the operating system. Subsequently, the quoting enclave signs the quote structure using the attestation key, and forwards it to the remote service provider and the Intel Attestation Service. The IAS verifies that the signature has been generated by a genuine non-revoked Intel SGX hardware, while the service provider compares the measurement value to a reference one, so as to ensure that the enclave code has not been tampered with during the loading process[11].

Ecosystem collaboration is at the heart of confidential computing

Confidential computing is an industry-wide effort that requires the cooperation of several stakeholders. On the hardware side, silicon providers have been investing considerable resources into maturing their TEE offerings. Just to cite a few, we have Intel SGX, Intel TDX, and AMD SEV on the X86 architecture; TrustZone and the upcoming ARM CCA for the ARM ecosystem; and Keystone for RISC-V architectures. Public cloud providers (PCPs for short) have been one of the main adopters of hardware trusted execution environments. In order to make running confidential workloads easy for their users, PCPs have been focusing on enabling a "shift and lift" approach, where entire VMs can run unchanged within the TEE, and are referred to as CVMs. What this means is that developers neither have to refactor their confidential applications nor rewrite them. What this also means is that the guest operating system needs to be optimised to support the user applications to leverage the platform's underlying hardware TEE capabilities, and to further protect the VM while it's booting, and when it's at rest.

Ubuntu confidential portfolio

This is exactly what Canonical has been focusing on. Thanks to a close collaboration with many major cloud providers, Ubuntu users can start their confidential computing journey today.

On Microsoft Azure, for example, it only takes a few clicks to enable and use Ubuntu 22.04 Confidential VMs (CVMs). They are part of the Microsoft Azure DCasv5/ECasv5 series that leverage the latest security extensions of the third generation of AMD CPUs, Secure Encrypted Virtualization-Secure Nested Paging (SEV-SNP). Ubuntu is the only Linux distribution supporting Azure Confidential VMs.

Similarly, on Google Cloud, you can already use Ubuntu confidential VMs that are also built on top of AMD SEV.

In order to drive the adoption of confidential computing with its users and customers, Canonical is an active member of the confidential computing consortium, a project community at the Linux Foundation that is focused on accelerating the adoption of confidential computing and driving cross-industry collaboration around relevant open source software, standards and tools.

How do Ubuntu AMD SEV Confidential VMs work?

Ubuntu CVMs achieve their strong security guarantees by securing your VMs throughout their entire lifecycle:

1. At run-time: Using AMD SEV-SNP, your VM's code and data are encrypted when they are being operated on in the system memory. The encryption leverages the newest AES-128 hardware encryption engine embedded in the CPU's memory controller. The encryption key is further protected and managed by the AMD Secure Processor.

2. At rest: Your entire workload is encrypted using Ubuntu-enhanced full disk encryption capabilities. The encryption key is itself stored encrypted in your VM's virtual disk. It's then bound to the virtual TPM (vTPM) associated with your instance. Finally, the vTPM is itself part of the guest VM address space, and enjoys the same run-time security guarantees provided by the AMD SEV-SNP extensions to the entire VM instance.

3. At boot time: Before booting the VM, the platform provides a hardware-rooted signed attestation which can be used to verify the OS, firmware and platform boot measurements.

Confidential computing as a part of your larger security strategy

While confidential computing aims to protect your workload against threats from other VMs, the cloud's privileges system software and its operators, it cannot protect you against software vulnerabilities that could potentially exist within your workload, even if it is running within a hardware-based TEE. Therefore, it is important that you have a sound and secure patch management policy that can keep your workload patched and up-to-date. It is also important for your deployment to adhere to industry best practices for hardening.

Security patch management

In a report published by Verizon 2022, only 25% of the scanned organisations were found to patch known vulnerabilities within two months of their public disclosure, but at the same time, exploiting known vulnerabilities is one of the most common causes of organisations being compromised in ransomware attacks and data breaches. While nobody can prevent zero-day attacks (where hackers take advantage of previously unknown software flaws), patching known vulnerabilities should be a fundamental part of every organisation's security strategy. It is good practice to scan systems regularly to check for known vulnerabilities, and at the same time also check for weak configuration settings. Triaging the results of these scans forms another part of the strategy.

Ubuntu and unattended upgrades Ubuntu has the ability to automatically apply security patches for vulnerabilities through the unattended-upgrades package, which checks for and applies security updates daily. This mechanism works for almost all packages within the Ubuntu ecosystem, except for the core system libraries and the Linux kernel.

Livepatch

In order to patch the Linux kernel without rebooting and causing downtime, Canonical offers Livepatch as part of an Ubuntu Pro subscription. Livepatch removes the need for unplanned maintenance windows for high and critical severity kernel vulnerabilities by patching the Linux kernel while the system runs.

Learn about Ubuntu Livepatch Service

Hardening and CIS benchmarks

Hardening a system is synonymous with reducing its attack surface: remove unnecessary software packages, lock down default values to the tightest possible settings and configure the system to only run what you explicitly require. This is something that needs to be applied with judgement based on the risks to the systems in question. It's very important to harden your production environments and critical systems, including anything that is internet-facing, but may be less relevant to development machines for instance. Regardless of the OS you are using, the best place to start when hardening systems is to look at CIS benchmarks for best practices. Given how comprehensive and detailed the CIS benchmarks are, a number of software vendors offer tools to automate their implementation. Canonical, for instance, has worked with CIS to provide an automated way to apply the benchmark rules to an Ubuntu system, as well as to audit and check a system to see how many of the benchmark rules are being adhered to. This is known as the Ubuntu Security Guide (USG).

Ubuntu Security Guide

USG is an easy to use tool for compliance and auditing on Ubuntu operating systems. USG encodes the hundreds of rules from the CIS benchmarks, allowing you to apply the benchmark by re-configuring and tweaking the system in one simple operation. USG can also audit a system to generate a list of the benchmark rules which are not being adhered to. This output is available in both human and machine readable formats. USG is available for 20.04 Focal Fossa LTS (with the latest 22.04 Jammy Jellyfish LTS version in development).

Learn more about security certification and hardening

A single subscription for security and compliance

Ubuntu users who want to complement confidential VMs with the above discussed hardening, patch management and livepatching capabilities, can rely on Ubuntu Pro, a comprehensive subscription for security and compliance.

The subscription expands Canonical's ten-year security coverage and optional technical support to an additional 23,000 packages beyond the main operating system. It is ideal for organisations looking to improve their security posture, not just for the Main repository of Ubuntu, but for thousands of open-source packages and toolchains.

Ubuntu Pro's coverage spans critical, high and selected medium CVEs for the kernel and thousands of applications and toolchains, including Ansible, Apache Tomcat, Apache Zookeeper, Docker, Nagios, Node.js, phpMyAdmin, Puppet, PowerDNS, Python, Redis, Rust, WordPress, and more.

Besides providing timely security patches, Ubuntu Pro includes tools for compliance management in regulated and audited environments. Ubuntu Security Guide enables best-in-class hardening and compliance standards such as CIS benchmarks and DISA-STIG profiles. Ubuntu Pro users can access FIPS-certified cryptographic packages necessary for all Federal Government agencies as well as organisations operating under compliance regimes like FedRAMP, HIPAA, and PCI-DSS.

Ubuntu Pro also includes Livepatch, and Landscape which can be used to facilitate system management and automated patching at scale.

Ubuntu Pro is available for every Ubuntu LTS from 16.04 LTS. It is already in production for large-scale customers offering global services. It is also available through our public cloud partners' marketplaces – AWS, Azure and Google Cloud.

Looking ahead

By leveraging hardware-rooted trusted execution environments, confidential computing aims to change the threat model of the public cloud. Its goal is to offer your workload strong integrity and confidentiality guarantees that can protect it against a potentially malicious cloud system software or a rogue cloud administrator. It achieves this by providing your workload with an isolated execution environment that benefits from run-time main memory encryption, and remote attestation capabilities that enable you to cryptographically verify the security claims of your public cloud provider.

If you are already using the public cloud, you can only benefit from running your workloads as confidential workloads instead. If you have security concerns that are preventing you from using the public cloud, the advances in confidential computing warrant that you re-evaluate your risk assessment, and reach the conclusion that best suits your organisation.

Ubuntu remains the most popular operating system in the cloud not just because of how it enables developers to create, but also because it enables them to do so securely. Ubuntu's advances in areas like confidential computing make it the clear choice for any organisation dealing with sensitive workloads. And Ubuntu Pro makes security even easier with features like FIPS-certification at launch, automated DISA-STIG/CIS auditing and hardening, and optional 24/7 support along with a 10-year lifetime security guarantee.

Learn more

At Canonical, we believe that confidential computing and privacy enhancing technologies will be the default way of doing computing in the future. This is why our confidential computing portfolio is free on all public clouds.
If you would like to know more about the Canonical approach to confidential computing and security at large, contact us.

Additional resources

- Watch our webinar to learn more about confidential computing
- Read our blog post for "What is confidential computing? A high-level explanation for CISOs"
- Read our blog post for "Confidential computing in public clouds: isolation and remote attestation explained"
- Start creating and using Ubuntu CVMs on Azure
- Is Linux Secure?
- Learn more about Google Cloud Confidential Computing
- Learn about Learn about how confidential computing can help you better utilize security-sensitive data within the financial-services industry
- Open source security: best practices for early detection and risk mitigation | webinar
- Ubuntu Pro | product page
- Ubuntu Pro | plans and pricing
- Buy Ubuntu Pro

References

[1] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. "Trusted Execution Environment: What It is, and What It is Not". In: Aug. 2015, pp. 57–64. doi: 10.1109/Trustcom.2015.357.

[2] https://docs.google.com/document/d/1w2p_YI4u85BTSBYrLO6ezcC8fpg84EU9N2YgKgOLWj8/edit#

[3] Coreboot. 2018. url: https://www.coreboot.org/vendors.html.

[4] Nathan D Dautenhahn. "Protection in commodity monolithic operating systems". PhD thesis. 2016. isbn: 978-1-267-82489-9.

[5] Moritz Lipp et al. "Meltdown: Reading Kernel Memory from User Space". In: 27th USENIX Security Symposium (USENIX Security 18). Baltimore, MD: USENIX Association, 2018, pp. 973–990. isbn: 978-1-931971-46-1. url: https://www.usenix.org/conference/usenixsecurity18/presentation/ lipp.

[6] Alexandra Dmitrienko et al. "Market-Driven Code Provisioning to Mobile Secure Hardware". In: Financial Cryptography and Data Security. Ed. by Rainer Böhme and Tatsuaki Okamoto. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 387–404. isbn: 978-3-662-47854-7

[7] Brian McGillion et al. "Open-TEE – An Open Virtual Trusted Execution Environment". In: Proceedings of the 2015 IEEE Trustcom/BigDataSE/ISPA - Volume 01. TRUSTCOM '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 400–407. isbn: 978-1-4673-7952-6. doi: 10.1109/Trustcom.2015.400. url: http://dx.doi.org/10.1109/Trustcom. 2015.400.

[8] Introduction to Secure Elements. Technical Report. Global Platform, 2018. url: https://globalplatform.org/wp-content/uploads/2018/05/Introductionto-Secure-Element-15May2018.pdf.

[9] Ekberg JanErik. "Securing Software Architectures for Trusted Processor Environments". PhD thesis. 2013. isbn: 978-952-60-3632-8.

[10] Intel® 64 and IA-32 Architectures Software Developer's Manual. Specifications. Intel, Oct. 2016. url: https://software.intel.com/en-us/articles/intelsdm.

[11] Victor Costan and Srinivas Devadas. "Intel SGX Explained". In: IACR Cryptology ePrint Archive 2016 (2016), p. 86.

Canonical