WP516 (v1.0.1) February 13, 2020

Isolate Security-Critical Applications on Zynq UltraScale+ Devices

Security across all markets is becoming more critical. Using a Trusted Execution Environment (TEE) on the Zynq® UltraScale+™ platform provides a major security advantage by isolating security-critical elements from the rest of the system.

ABSTRACT

Implementing a TEE on the Zynq® UltraScale+™ platform (RFSoCs and MPSoCs) greatly reduces the attack surface of security-critical applications.

Explore this white paper to find out

- What a TEE is
- How the requirements for a TEE are easily met on the Zynq® UltraScale+™ platform
- Why a TEE is needed, even if hypervisors are used
- An example architecture of Prove & Run's ProvenCore TEE [Ref 1] running on the Zynq® UltraScale+™ platform
- Real-world TEE usage examples in automotive and data center applications

Introduction

What is a Trusted Execution Environment?

A trusted execution environment (TEE), also known as a secure execution environment (SEE), is an environment dedicated to running security-critical tasks. The TEE includes both trusted hardware and trusted software, referred to as the *trusted world*. The *untrusted* or *non-secure* world refers to both untrusted hardware and untrusted software. Because the TEE is isolated from the untrusted world, an extremely high level of security is provided. Normally, a TEE runs alongside the untrusted world on the same processor or system-on-a-chip (SoC) and provides trusted services on behalf of the untrusted world.

Compared to external security modules such as the trusted platform module (TPM) or smart cards, which encapsulate security inside a highly secure separate physical device, a TEE has much higher computational performance (gigahertz vs. megahertz), access to a much larger set of RAM (gigabytes vs. kilobytes), and access to a much richer set of peripherals (gigabit Ethernet, programmable logic (PL), hardware accelerators, etc.). The resources to which the TEE has access are limited only by the processor on which the TEE is running. Additionally, external security modules still face a security vulnerability because they are normally controlled and accessed from untrusted systems, making them prone to attack. For example, if an external module needs a password to access a certain security feature, that password goes through the untrusted system, exposing the password to numerous types of attacks. On the other hand, using a TEE with a trusted input device—fingerprint reader, pin pad, etc.—the password is only read in and processed by trusted hardware and software, minimizing the likelihood of any possible attacks, and greatly minimizing the attack surface. This simple example shows why a TEE provides greater security through isolation, and is illustrated in Figure 1.

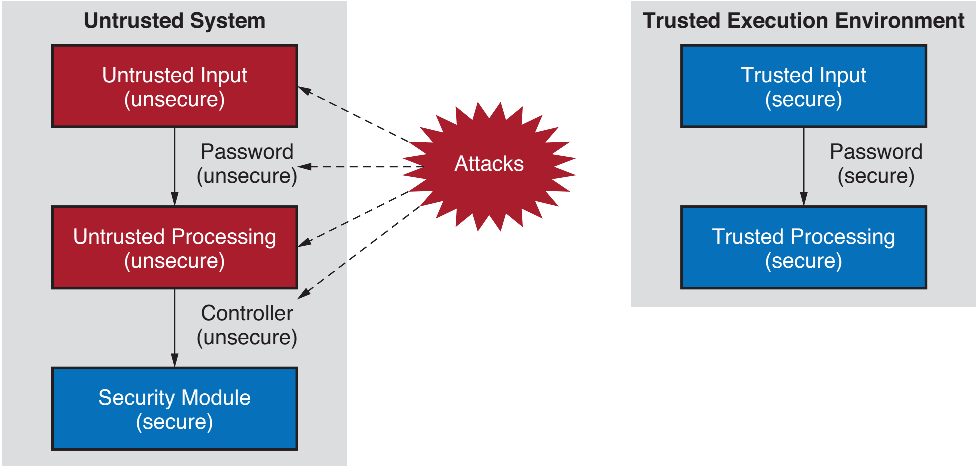


Figure 1: **Attack Surface Reduction of a TEE Compared to a Typical Untrusted System**

To run any security-critical software in the TEE, a trusted OS is used to handle the scheduling and operation of the secure software. The trusted OS is different than a typical OS—Linux, Windows, etc.—because the code base is extremely small, on the order of kilobytes. The small code base of the trusted OS is important because the entire operating system can be inspected and tested against any known vulnerabilities. This greatly minimizes the trusted OS's attack surface while providing a mechanism of software isolation from any untrusted software.

Trusted applications (TA) run inside the trusted OS and provide any security-critical functionality needed by a system. Just like untrusted applications, trusted applications are portable across multiple architectures running the same secure OS, meaning that TA developers do not need to rewrite their TA for every platform they want to use. In addition to the trusted OS providing isolation from any untrusted software, using a trusted OS also provides further isolation by isolating trusted applications from each other.

There is no reason that a TA with access to a particular AES encryption key needs access to a particular RSA key used by *another* TA. Even though TA to TA isolation is enforced by the secure OS, a TA still has the ability to call another TA if needed, which allows for keeping trusted application code small and modular.

TEE Requirements

This section of the white paper describes all the architecture level components required for running a trusted execution environment on a system and covers:

- Hardware Enforced Isolation
- Software Enforced Isolation
- Secure Monitor
- Shared Memory
- Trusted Boot

Any single component by itself does not facilitate the use of a TEE, but these components must be used together.

Hardware Enforced Isolation

One mechanism for isolating the trusted world from the untrusted world is by using physically separate processors. This is similar to using the external hardware module mentioned above, but different, because both processors are usually in the high-performance category. The drawback to using this type of architecture is that the use of multiple processors increases the overall cost, and it uses much more board space. On multi-core processors, a single core can be dedicated to the trusted world, and another core can be dedicated to the untrusted world. This type of architecture saves cost and board space compared to the multi-processor architecture. However, the major drawback to both these architectures is that they still need access to external peripherals such as dynamic random access memory (DRAM), Ethernet, etc. Physically separate peripherals can be used at an increased cost and board size to achieve physical isolation between the two worlds; otherwise, the peripherals have to be shared, and the isolation between peripherals can only be achieved through a weaker software-isolated mechanism—not strong enough for a highly secure TEE.

To use a single processor with a single set of peripherals and still achieve hardware isolation, the Xilinx® Zynq® UltraScale+™ platform uses Arm® TrustZone technology [Ref 2] as one form of hardware isolation. TrustZone not only provides hardware-enforced isolation between processors and their individual cores, but enforces hardware isolation across the entire platform, providing system-wide security. When TrustZone is enabled, all AXI transactions, memory locations, peripherals, interrupt controllers, hardware accelerators, caches, and processor execution states are automatically stored in physical registers with a single bit indicating whether the resource is accessible from either the secure or non-secure world. When the bit is set, the resource is non-secure; when the bit is clear, the resource is secure.

The trusted world has access to the untrusted world, but not the other way around. If a non-secure resource tries to gain access to a secure resource, a security exception is thrown, immediately stopping any potential threats at the physical hardware level. For example, a processor executing in the non-secure state is denied access to any secure memory. On the other hand, the secure world might need access to untrusted resources for register configuration or for reading untrusted memory buffers.

Another important feature that TrustZone entails is the ability to use the same resource in both a secure state as well as a non-secure state. This is most commonly implemented in processors where some processing occurs in the secure state, and then the processor is switched over to execute in the non-secure state. Since the processor's caches and other associated memories are TrustZone aware, a context switch between the trusted world and untrusted world is extremely fast. Caches and other associated memories do not need to get flushed and reloaded when switching between worlds.

Isolation Enhancements on the Zynq UltraScale+ Platform

On the Zynq UltraScale+ platform, hardware-enforced isolation is further enhanced by Xilinx's Peripheral Protection Unit (XPPU) [Ref 3] and Xilinx's Memory Protection Unit (XMPU) [Ref 3]. The use of the XMPU and XPPU on the Zynq UltraScale+ platform allows the system to be isolated even further, since these protection units use the incoming master AXI identification (ID) to filter any request in addition to checking the TrustZone status of the incoming transaction. With this additional enhancement, not only can the Zynq UltraScale+ platform be split in two worlds—secure and non-secure—but it can be split into numerous domains. The XMPU is specifically used for isolating the DDR, on-chip memory (OCM), and any full-power domain (FPD) [Ref 3] peripherals. The XPPU is specifically used for isolating the low-power domain (LPD) [Ref 3] peripherals, QSPI, inter-processor interrupt (IPI) message buffers, and the Arm® Cortex®-R5 processor's tightly coupled memory (TCM) banks. Extensive details of how to set up and use TrustZone, the XMPU, and the XPPU are outlined in XAPP1320 [Ref 4]. Proper isolation configuration of a system, like the Zynq UltraScale+ platform, is an important requirement of a TEE.

With respect to the master IDs, all four cores of the Arm Cortex-A53 processor are grouped together into one master ID instead of four separate IDs, meaning that all four cores can operate only in the secure or non-secure mode at any one time. On the other hand, each of the Arm Cortex-R5 processors is assigned a unique ID while operating in split mode, and a single ID is assigned while operating in lock-step mode. Multiple IDs are assigned to the PL and can be designed into IP in the PL so that hardware isolation is not limited to just the processing units.

The Arm Cortex-A53 cores implement a memory management unit (MMU) [Ref 5] to provide virtual addressing while the Arm Cortex-R5 processor implements a memory protection unit (MPU) [Ref 6] to control access to and from L1-cache as well as external memory. To extend the functionality of the MMU to the SoC level, the Zynq UltraScale+ platform implements a system MMU (SMMU) [Ref 3]. Additionally, Advanced eXtensible Interface (AXI) and Advanced Peripheral Bus (APB) isolation blocks (AIB) are implemented throughout the Zynq UltraScale+ platform to prevent undesired access by an AXI/APB master to an AXI/APB slave. The device also implements an AXI timeout block (ATB) that prevents an AXI master from hanging when no response is received from an AXI slave. ATB is not necessarily an additional form of isolation, but it is an important Zynq UltraScale+ platform security feature, because an AXI master can recover if one of its AXI slaves locks up, preventing a simple denial of service (DoS) attack.

Software Enforced Isolation and the Secure Monitor

Just as TrustZone is used as a hardware-enforced mechanism to isolate a system into two separate worlds, an additional software-enforced mechanism called *exception levels*[1] is present on Armv8-A processors, such as the Zynq UltraScale+ platform Arm Cortex-A53 processors [Ref 7].

There are non-secure exception levels (EL) when operating in the untrusted world and secure exception levels (SEL) when operating in the trusted world. Table 1 summarizes the exception levels and their normal use case. Notice that there is no SEL2 when operating in the trusted world, and that EL3 resides in the secure world, even though the name is not specified as SEL3.

Table 1: **Arm Exception Level Summary**

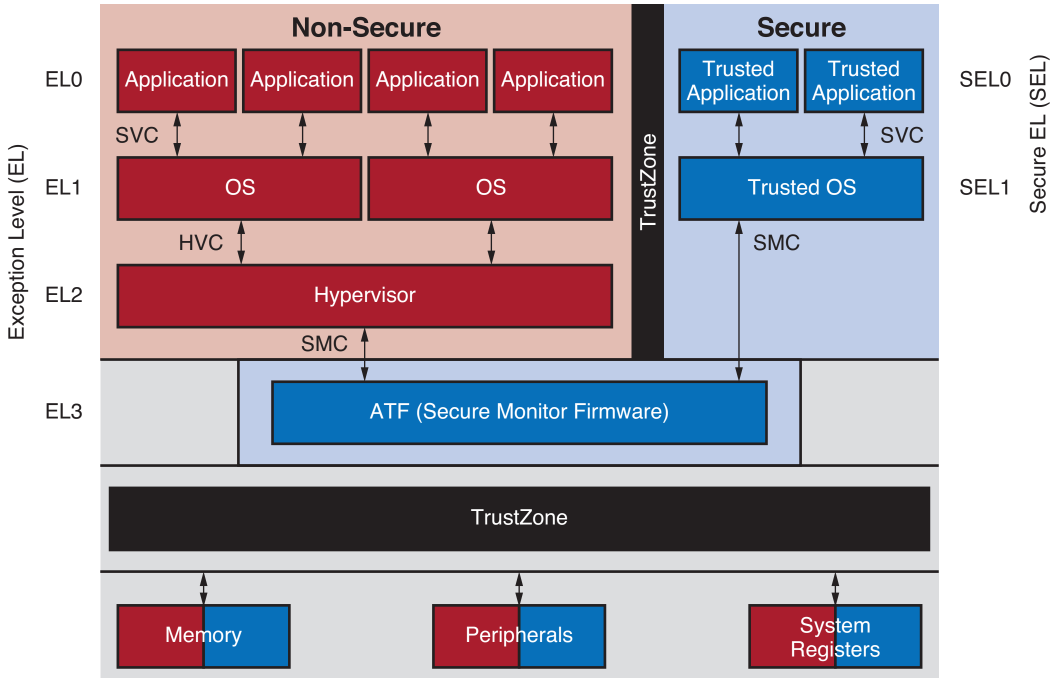
EL	Untrusted World Use Case	Trusted World Use Case	SEL
EL0	Execution of user applications	Execution of trusted applications	SEL0
EL1	Execution of an operating system	Execution of a trusted operating system	SEL1
EL2	Execution of a hypervisor	---	---
EL3	Used for switching between the non-secure and secure worlds		

For a processor to switch from executing in the untrusted world to the trusted world, a mechanism called the *secure monitor* handles this switching. In Arm systems, the secure monitor can only be executed out of EL3. This naming convention might seem to suggest that EL3 is not secure; however, EL3 is only accessible by issuing the secure monitor call (SMC) instruction [Ref 7] while the processor is operating in EL1, SEL1, or EL2. Arm has its own open-source secure monitor software called Arm Trusted Firmware (ATF) [Ref 8], and Xilinx supports the version of ATF for use with the Zynq UltraScale+ platform.

Figure 2 summarizes the use of hardware and software isolation in a TEE architecture highlighting the use of TrustZone and Arm exception levels.

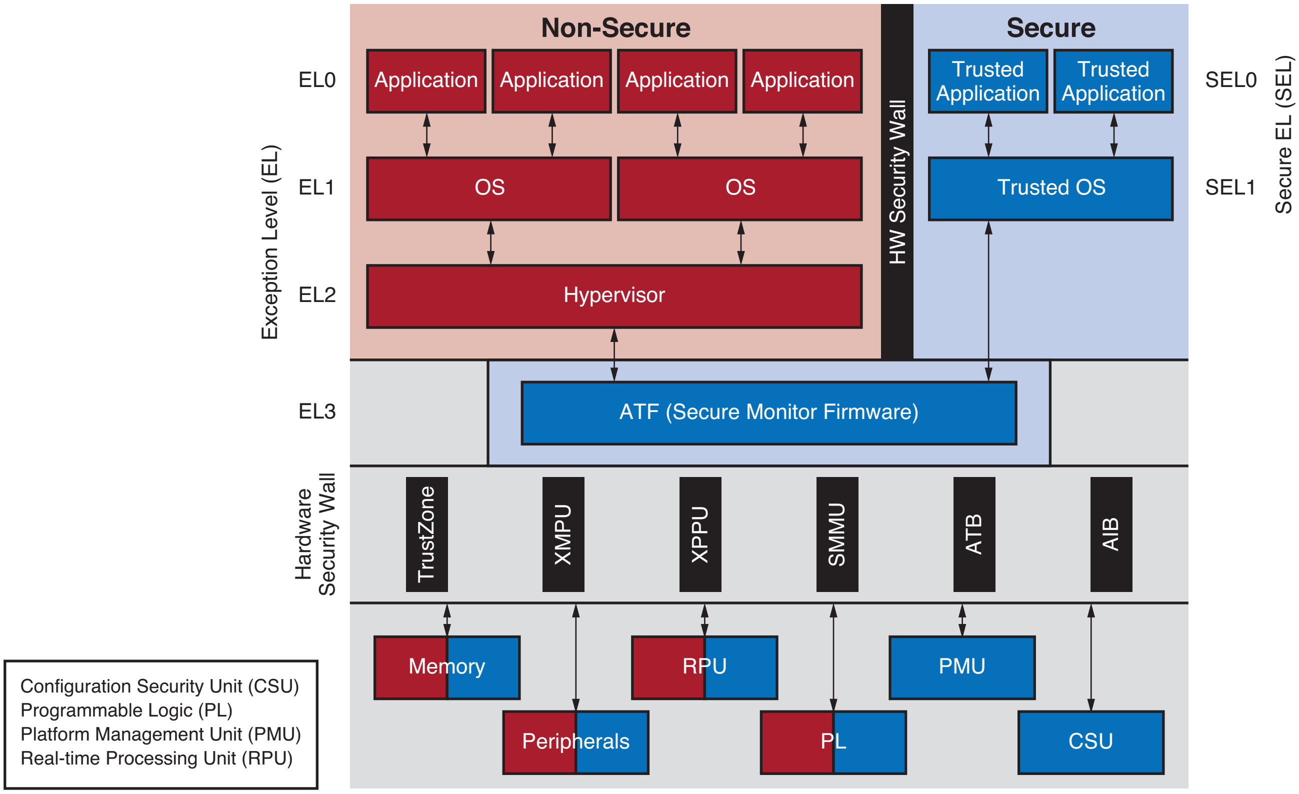
For comparison, Figure 3 illustrates an enhanced TEE system architecture, showing the additional layers of hardware isolation mechanisms in place, as implemented in the Zynq UltraScale+ platform. The architecture view is as seen from the Arm Cortex-A53 processors; for simplicity, the RPU and PL are treated as peripherals of the processor.

[1] The 32-bit software enforcement mechanism is called processor mode for the 32-bit operation of the Arm Cortex-A53 and Arm Cortex-R5 processors, described in Arm's technical documents [Ref 7].



WP516_02_092219

Figure 2: **A Generic Armv8-A TEE Architecture with Hardware Isolation through TrustZone Software Isolated through Arm Exception Levels**



WP516_03_092219

Figure 3: **Zynq UltraScale+ Platform Armv8-A TEE Architecture with Enhanced Isolation**

The key to switching between the non-secure world and the secure world is the SCR_EL3 register [Ref 9], which can only be accessed while a processor is operating at the EL3 level. This is why the secure monitor is required to operate out of EL3. SCR stands for Secure Control Register; bits 3 through 0 of the register are shown in Table 2.

The most important bit in this register is the NS bit. When this bit is set, a processor operates in the untrusted world. When this bit gets cleared, the processor operates in the trusted world. To change worlds, a processor enters the secure monitor software via the SMC instruction, changes the NS bit, and then returns from the secure monitor to operate in either the trusted or untrusted world.

Table 2: **SCR_EL3 Bit Assignments [3:0]**

Bit	Name	Function
[3]	EA	**EA External Abort and SError Interrupt Routing**. This bit controls which mode takes external aborts. The possible values are: **0:** External Aborts and SError Interrupts while executing at exception levels other than EL3 are not taken in EL3. This is the reset value. **1:** External Aborts and SError Interrupts while executing at all exception levels are taken in EL3.
[2]	FIQ	**Physical FIQ Routing**. The possible values are: **0:** Physical FIQ while executing at exception levels other than EL3 are not taken in EL3. This is the reset value. **1:** Physical FIQ while executing at all exception levels are taken in EL3.
[1]	IRQ	**Physical IRQ Routing**. The possible values are: **0:** Physical IRQ while executing at exception levels other than EL3 are not taken in EL3. **1:** Physical IRQ while executing at all exception levels are taken in EL3.
[0]	NS	**Non-secure Bit**. The possible values are: **0:** EL0 and EL1 are in Secure state, memory accesses from those exception levels can access Secure memory. This is the reset value. **1:** EL0 and EL1 are in Non-secure state, memory accesses from those exception levels cannot access Secure memory.

Three additional bit fields in the SCR_EL3 register are important: fast interrupt request (FIQ), interrupt request (IRQ), and external abort (EA) bits. These bits indicate whether interrupts are routed through the secure monitor or if they are routed through the hypervisor or operating system, depending on the processor's configuration. Normally, FIQs are reserved for secure interrupts and IRQs are reserved for non-secure interrupts. Interrupt routing is also controlled by the GICv2 registers in the Zynq UltraScale+ platform [Ref 10], and access to the GICv2 registers is TrustZone-aware so that the interrupt registers can only be modified in the secure world. Passing interrupts through the secure monitor code in EL3 not only allows a flexible interrupt scheme, but ensures either secure or non-secure interrupts are handled in a timely manner. For example, if operating in the non-secure world, secure interrupts can route through the secure monitor and get handled immediately instead of being handled when the processor switches to the trusted world. Passing non-secure interrupts through the secure monitor allows checking of interrupt validity and possibly stopping any malicious interrupts from disrupting a system. These are just simple examples of interrupt routing, which is ultimately handled by the system architecture.

Shared Memory

If a trusted application is to provide trusted services to the non-secure world, a mechanism for passing data back and forth is needed as well as TEE access to the non-secure memory. Non-secure memory access by a TEE is already accomplished on a TrustZone-aware system like the Zynq UltraScale+ platform. The most common way to pass data back and forth between the trusted and untrusted worlds is to designate a portion of memory for sharing. When operating in the non-secure world, memory can be loaded, a TA called, and then read back when the TA is complete. While in the secure world, the secure OS or TA can place a copy of the memory into a portion of secure memory, operate directly on the non-secure memory, or change the state of the shared memory from non-secure to secure and then back to non-secure. Memory sharing operations are completely dependent on the TEE architecture.

Trusted Boot

For a TEE to operate in a secure manner, the TEE must be loaded in a secure manner; otherwise, any operations in the TEE cannot be considered secure. Xilinx provides all the necessary tools [Ref 11], and the Zynq UltraScale+ platform supports both authenticated and/or encrypted boot modes [Ref 3] [Ref 11].

TEE Compared to a Hypervisor

Background

There are two types of hypervisors:

- Type 1 hypervisors execute directly on the hardware
- Type 2 hypervisors execute as applications on top of an OS

In this white paper, only Type 1 hypervisors are considered, as the security of Type 2 hypervisors is highly dependent on the underlying OS they run on.

A *hypervisor* is a piece of software running at EL2 that emulates hardware systems and makes them available as virtual machines (VMs). As hypervisors can usually execute several VMs in parallel by allocating processor time and memory to each VM, hypervisors sometimes provide VM-to-VM communication, either by emulating traditional communication peripherals or by implementing a custom interface.

Hypervisors and TEEs both enable independent execution environments, VMs, and secure applications, respectively. This implies that they both manage the allocation of resources. However, they differ greatly in the services they provide and in the interfaces they present to their execution environments. Hypervisors expose the low-level interfaces of the hardware systems they emulate, unlike OSes that expose high-level interfaces for application development. To develop applications on a VM, the applications are installed on the VM's OS as needed.

Security Functions Benefit from a Secure OS

Hardware interfaces are very low-level, and application developers usually rely on an OS to provide high-level services such as control of interrupts, memory management, and inter-process communication. Applications can be directly developed using low-level hardware, but this involves dealing with the complexity of the hardware interfaces. In the case of security functions, implementing them in bare-metal amounts to implementing high-level abstractions directly inside the security functions for every security function required in a system. Aside from the fact that this is unrealistically difficult, especially in the case of SoCs, developing the high-level abstractions required by each security function is prone to error, which greatly increases a system's attack surface.

Hypervisors Do Not Replace the Use of a TEE

Even with secure hypervisors such as ProvenVisor [Ref 1], a hypervisor developed using deductive formal methods, there is no secure way of implementing security functions on the hypervisor itself. Developing security functions on individual bare-metal VMs to mitigate non-secure OS issues is possible, but the security function still has all the issues described above. The correct implementation of security functions depends on the correct implementation of the underlying OS, and developing secure applications on a non-secure OS in one of the VMs makes the security function vulnerable. The only viable solution is to resort to a TEE to host the security functions in a secure manner. However, a TEE does not replace the use of a hypervisor if virtualization needs to be implemented. It should be noted that a TEE can be used with or without the use a hypervisor.

ProvenCore TEE Running on the Zynq UltraScale+ Platform

Xilinx and Prove & Run have partnered together to port the EAL7 certified ProvenCore TEE [Ref 1] onto Zynq UltraScale+ devices. This enables the device to run a diverse set of security applications while reducing possible attacks. ProvenCore running on the Zynq UltraScale+ platform is shown in Figure 4. As with every port of ProvenCore, ProvenRun carefully designed the secure boot on the device. During boot, the first-stage boot loader (FSBL) loads ATF, which then loads ProvenCore in a secure mode. After the initial setup of ProvenCore, ProvenCore returns to ATF, which then boots Linux by launching U-Boot in non-secure mode.

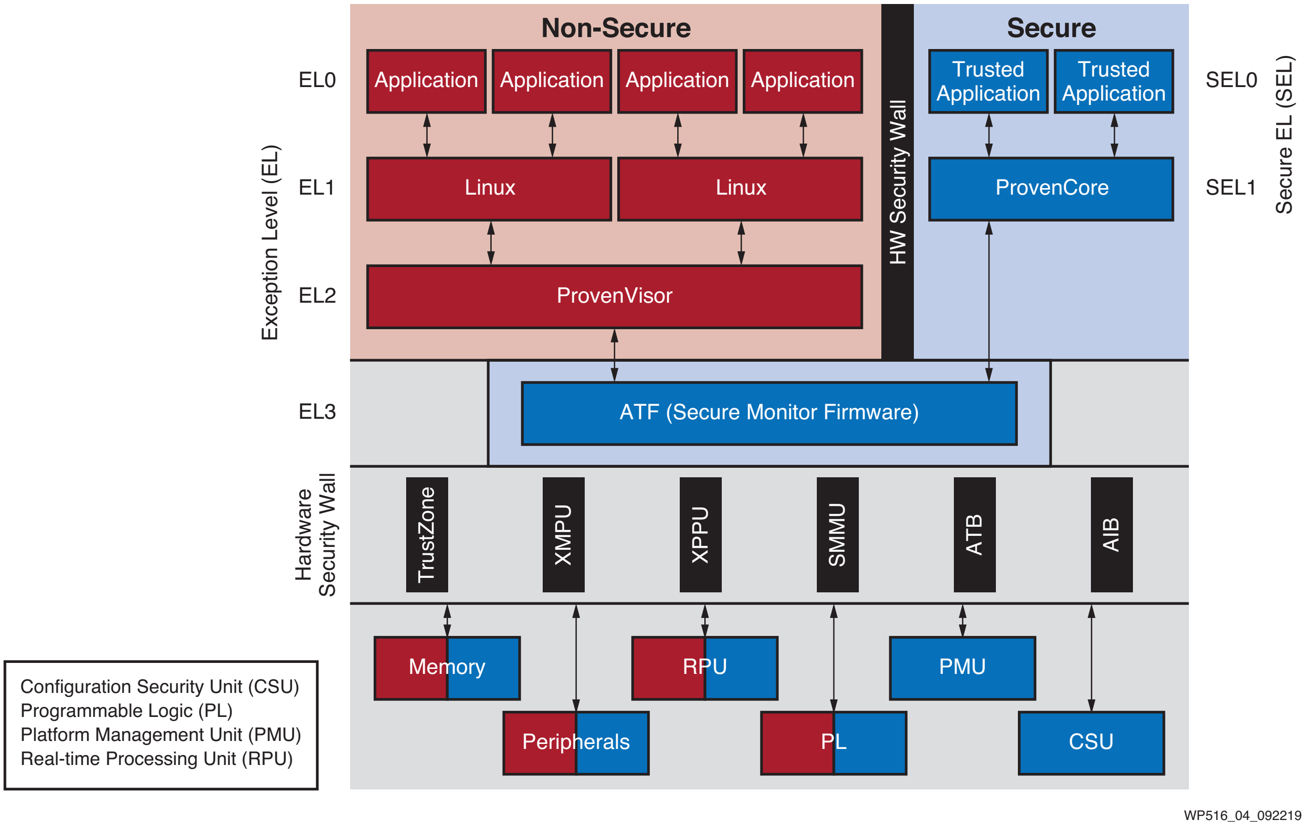


Figure 4: **ProvenCore Running on Zynq UltraScale+ Platform**

ProvenCore embeds a security driver that sets up security rules for peripheral access. On the Zynq UltraScale+ platform, this driver configures Arm TrustZone, the XMPU, and the XPPU to match the given security policy. The XMPU is used to guarantee that only ProvenCore can access its secure memory area. The XPPU is used to reserve either completely, or temporarily, at ProvenCore's request, peripherals that the secure environment would like to use. The eMMC controller is dedicated to secure storage in the secure world, and the Ethernet controller is completely operated from the secure world. If Linux crashes or gets corrupted, ProvenCore's secure peripherals continue to operate as normal and recover the system. ProvenCore also relies on the Zynq UltraScale+ platform's cryptographic cores for accelerating cryptographic functionality.

Communication between Linux and ProvenCore is provided by the Linux TrustZone driver and enables communication between a Linux user space application and trusted application running inside of ProvenCore. The driver shares a common DRAM area to exchange requests and responses between the secure and non-secure world. Every connection through the driver between a user space application and a TA is known as a *session* in Prove & Run's TEE terminology. Each session has a dedicated shared memory area and establishes its own communication protocol. World change requests are sent using two dedicated SMC calls and are dispatched using two reserved software-generated interrupts managed by the Secure Monitor. One SMC call is required to switch from the untrusted world, normally Linux, into ProvenCore. The other SMC call is used to switch back from ProvenCore into the untrusted world. In addition to receiving SMC calls from the untrusted world to switch into the trusted world, ProvenCore implements a user-configurable timer so that ProvenCore can be run periodically without being called by the untrusted world. Most TEEs are scheduled to run when time is allocated by the non-secure OS.

Since the PL of the Zynq UltraScale+ platform can propagate the secure status of a transaction through the TrustZone bit in the interconnect IP, ProvenCore also manages peripherals implemented in the PL through the combination of a driver running inside of ProvenCore as well as a Prove & Run hardware block running inside of the PL. This allows Prove & Run's TEE architecture to ban any non-secure access to any secure hardware running inside of the PL.

TEE Usage Examples

Increasing Security in Automotive

Increasingly, cars are connected to the outside environment. Many car manufacturers are opting for security architectures with multiple security domains that are compartmentalized using either hardware virtualization or multiple hardware devices. Some of these domains manage communications with untrusted environments and therefore are at the forefront when remote attacks occur. In some automotive architectures, an intrusion protection system (IPS) is deployed which scrutinizes all incoming traffic to detect possible attacks on the different communication layers. Unfortunately, communication stacks are complex and cannot be fully free from bugs. Attackers use these vulnerabilities coupled with additional vulnerabilities on the underlying non-secure OS, which the IPS is running on, to take control. From there, attackers can circumvent all security functions intended to block malicious traffic from reaching the safety-critical domains of the car.

However, if the IPS runs on a secure OS, most of the remote attack risk can be mitigated. The secure OS alone has a much smaller attack surface than a typical non-secure OS, which mitigates the secure OS from being compromised. While the IPS operates in the secure world, the IPS can perform re-encapsulation of messages for the different layers of the communication protocol stack. This prevents attacks that use vulnerabilities in the implementation of communication stacks and decreases the attack surface even further. Additionally, an initial check of the conformity of the message’s payload can be implemented. In the end, messages from the secure OS that are transmitted to applications running on a non-secure OS have a reduced probability of being compromised.

In this example, using a secure hypervisor does not, on its own, solve the issue. A hypervisor architecture can dedicate a VM for hosting the IPS, but if the OS running on this VM can be compromised by incoming messages, then the OS is unable to mitigate being attacked. Messages transmitted from a VM containing the IPS to the other parts of the car can be as malicious as those coming directly from the outside. Therefore, a secure OS is necessary for the VM on which the IPS is running on.

Increasing Security in a Data Center

Applying algorithms for medical diagnostics can be computationally intensive, which motivates the use of cloud computing for offering these services. However, patient data, which is often very valuable to hackers, must be sent to a data center to be analyzed. Encrypted communication and storage help against data theft and usually rely on hardware security modules (HSM) for key management and cryptographic computations.

In the typical architecture, as shown in Figure 5, a Linux server is in charge of establishing secure communication with the client by using the cryptographic services of an HSM. The server then performs the computations on the unencrypted patient's data, which can involve using FPGA peripherals for increased speed. Afterwards, the server sends the data back to the client using the secure communication channel. The Linux server now has access to unencrypted patient data and diagnosis. A hacker can install back doors in the server or exploit common vulnerabilities to steal the data. A TEE, however, can run a secure OS like ProvenCore and keep all confidential patient data out of the non-secure Linux server. While inside the TEE, the patient data is securely decrypted, confidentially processed, securely re-encrypted, and then sent back safely through the Linux server without compromise. Moreover, any high-performance processing can be securely computed inside of the TEE because the PL on the Zynq UltraScale+ platform is TrustZone aware.

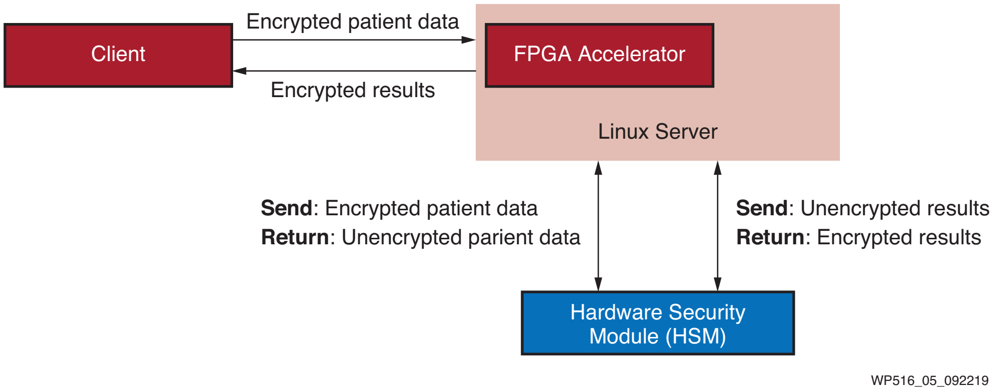

Client
Encrypted patient data
Encrypted results
FPGA Accelerator
Linux Server
Send: Encrypted patient data
Return: Unencrypted parient data
Send: Unencrypted results
Return: Encrypted results
Hardware Security Module (HSM)
WP516_05_092219

Figure 5: **A Typical Data Center Architecture Used for Analyzing Medical Data**

Conclusion

This white paper explored how a TEE running on the Zynq UltraScale+ platform greatly reduces the attack surface of security-critical applications and greatly increases system-wide security. By taking advantage of all the security features implemented on the Zynq UltraScale+ platform—including TrustZone, XMPU, XPPU, SMMU, AIB, Arm Exception Levels, Xilinx ATF, and trusted boot—a strong trusted execution environment can be created using only a single SoC. This white paper showed why a hypervisor still requires the use of a TEE to isolate security critical tasks, since those tasks might be running on potentially non-secure operating systems. The TEE architecture designed by Prove & Run showed how their secure OS (ProvenCore) was implemented using the Zynq UltraScale+ platform. Not only can a TEE be used across all markets, but an automotive application and a data center application were described in detail to show how a TEE greatly increases security in both applications.

Note: This white paper has been written in collaboration with our TEE partner Prove & Run.

References

1. Prove & Run website
2. Arm Technologies: TrustZone for Cortex-A, *SoC and CPU System-Wide Approach to Security*
3. Xilinx User Guide UG1085, *Zynq UltraScale+ Device Technical Reference Manual*
4. Xilinx Application Note XAPP1320, *Isolation Methods in Zynq UltraScale+ s*
5. Arm System Memory Management Unit Architecture Specification: *SMMU Architecture 2.0*
6. Arm Technical Reference Manual: *Cortex-R5 and Cortex-R5F*
7. Arm: *Fundamentals of ARMv8-A*
8. Arm: at GitHub, *ARM Trusted Firmware-A* (read-only mirror)
9. Arm: *Cortex®-A53 MPCore Processor Technical Reference Manual, Rev. r0p4*
10. Arm: *Generic Interrupt Controller - Architecture version 2.0 - Architecture Specification*
11. Xilinx User Guide UG1137, *Zynq UltraScale+ Software Developer Guide*

Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
02/13/2020	1.0.1	Typographical edit.
12/12/2019	1.0	Initial Xilinx release.

Disclaimer

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at http://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at http://www.xilinx.com/legal.htm#tos.

Automotive Applications Disclaimer

XILINX PRODUCTS ARE NOT DESIGNED OR INTENDED TO BE FAIL-SAFE, OR FOR USE IN ANY APPLICATION REQUIRING FAIL-SAFE PERFORMANCE, SUCH AS APPLICATIONS RELATED TO: (I) THE DEPLOYMENT OF AIRBAGS, (II) CONTROL OF A VEHICLE, UNLESS THERE IS A FAIL-SAFE OR REDUNDANCY FEATURE (WHICH DOES NOT INCLUDE USE OF SOFTWARE IN THE XILINX DEVICE TO IMPLEMENT THE REDUNDANCY) AND A WARNING SIGNAL UPON FAILURE TO THE OPERATOR, OR (III) USES THAT COULD LEAD TO DEATH OR PERSONAL INJURY. CUSTOMER ASSUMES THE SOLE RISK AND LIABILITY OF ANY USE OF XILINX PRODUCTS IN SUCH APPLICATIONS.