**Introduction to TEE of ARMv8**

Increasing popularity of connected devices in recent years has led devices manufacturers to deal with security issues in a more serious way than before. In order to address these issues appropriately, a specification has emerged to define a way to ensure the integrity and confidentiality of data running in the entity implementing this specification.

## Trusted Execution Environment

A Trusted Execution Environment (TEE) is a secure area inside a main processor. It runs in parallel of the operating system, in an isolated environment. It guarantees that the code and data loaded in the TEE are protected with respect to confidentiality and integrity. This alongside-system is intended to be more secure than the classic system (called REE or Rich Execution Environment) by using both hardware and software to protect data and code.

Trusted applications running in a TEE have access to the full power of a device's main processor and memory, whereas hardware isolation protects these components from user installed applications running in the main operating system. Software and cryptographic isolations inside the TEE protect the different contained trusted applications from each other.

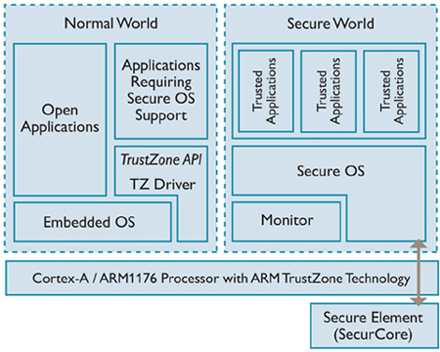
This starts a series of two blogposts discussing hardware technologies that can be used to support TEE implementations:

* TrustZone from ARM
* SGX from Intel

As suggested by the title, this blogpost tells you more about TrustZone.

ARM processors with TrustZone implement architectural Security Extensions in which each of the physical processor cores provides two virtual cores, one being considered non-secure, and called Non Secure World, the other being considered Secure and called Secure World, and a mechanism to context switch between the two, known as the monitor mode.

A schema from ARM:



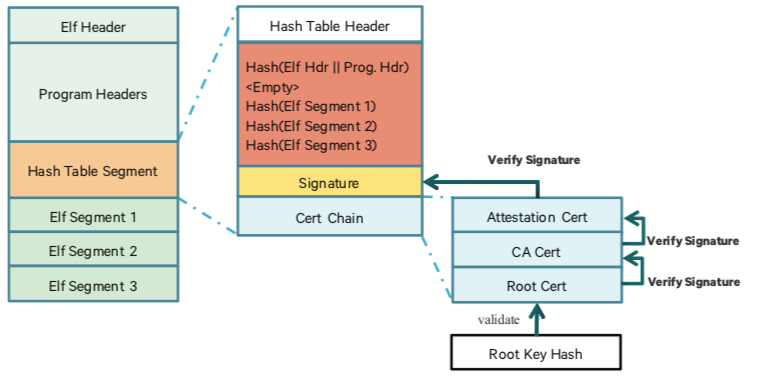
As illustrated by this figure, TrustZone consist in a monitor, an optional OS and optional applications, all running in Secure World. A Trustzone implementation could be all those components like on the Qualcomm or Trustonic implementations, or only a Monitor as the Nintendo Switch implementation does.

Implementing a TrustZone OS provides a more flexible model for adding trusted functionality which are meant to provide additional secure service to the Normal World.

These features are available as signed third-party applications (called trustlet), and are securely loaded and executed in Secure World by the operating system running in TrustZone (the SecureOS).

One of these secure loading features (namely the Qualcomm one) was fully [retro engineered by Gal Beniamini](http://bits-please.blogspot.com/2016/04/exploring-qualcomms-secure-execution.html), before Qualcomm [made it pubic](https://www.qualcomm.com/media/documents/files/secure-boot-and-image-authentication-technical-overview.pdf).

The trustlet integrity checking process is relatively standard and consists of a hash table, each hash represents the hash of a segment of the ELF binary. This hash table is then signed by the trustlet issuer, and this signature can be verified through a certificate chain placed directly after the signature. This chain of trust can be validated thanks to the root certificate of which a SHA256 is placed in a fuse (QFuse), ensuring its integrity.



Source: Qualcomm

TrustZone is used for many purposes, including DRM, accessing platform hardware features such as stored RSA public key hash in eFuse, Hardware Credential Storage, Secure Boot, Secure Element Emulation, etc.

## Secure World vs Normal World - Context Switch Mechanisms

The world in which the processor is currently running can be determined by the Non-Secure bit available in the Secure Configuration Register. The mechanisms by which the physical processor can enter monitor mode from the Non-Secure World can be triggered by executing a dedicated instruction called Secure Monitor Call (SMC) and it could be seen as an exception to the monitor mode software. Moreover, the main memory is also flagged with the Non-Secure bit, allowing memory isolation between Secure and Non-Secure worlds.

### Exception handling in ARMv7

Exception handling in ARMv7 requires the Secure World to register the address of a vector to which the processor will jump to when an SMC instruction is encountered. This handler is located at the offset 0x8 from the address stored in the Monitor Vector Base Address Register (MVBAR) in ARMv7.

### Exception handling in ARMv8

In ARMv8, a new exception model has been introduced which defines the concept of exception levels. An exception level determines the privilege level (PL0 to PL3) at which software components run and which processor modes (non-secure and secure) shall be used to run it. Execution at ELn corresponds to privilege PLn and, the greater the n is, the more privileges an execution level has. When an exception occurs, the processor branches to an exception vector table and runs the corresponding handler. In ARMv8, each exception level has its own exception vector table.

### Privilege levels

In ARMv5, User modes are known as unprivileged modes, while the following modes are known as privileged modes:

* FIQ.
* IRQ.
* Supervisor.
* Abort.
* Undefined.
* System.

In ARMv7-A with the Virtualization implementation, the privileged model is different from that in ARMv5. In Non-secure state, ARMv7-A has three privilege levels, PL0, PL1, and PL2:

PL0

The privilege level of application software that executes in User mode. Software executed in User mode is described as unprivileged software. This software cannot access some features of the architecture. In particular, it cannot change many of the configuration settings.

Software executing at PL0 can make only unprivileged memory accesses.

PL1

Software execution in all modes, other than User mode and Hyp mode, is at PL1. Normally, operating system software executes at PL1.

PL1 modes refer to all modes other than User mode and Hyp mode.

An Operating System is expected to execute across all PL1 modes, with its applications executing at PL0 (User Mode).

PL2

Hyp mode is normally used by a Hypervisor and can switch between Guest Operating Systems that execute at PL1.

If Virtualization Extensions are implemented, a Hypervisor executes at PL2 (Hyp mode). A Hypervisor controls and enables multiple Operating Systems to co-exist and execute on the same processor system.

These privilege levels are separate from the TrustZone Secure and Non-secure settings. [Figure 3.20](http://infocenter.arm.com/help/topic/com.arm.doc.dai0425/ch03s10s01.html" \l "CEGEJIGD) shows processor modes and their corresponding privilege levels.

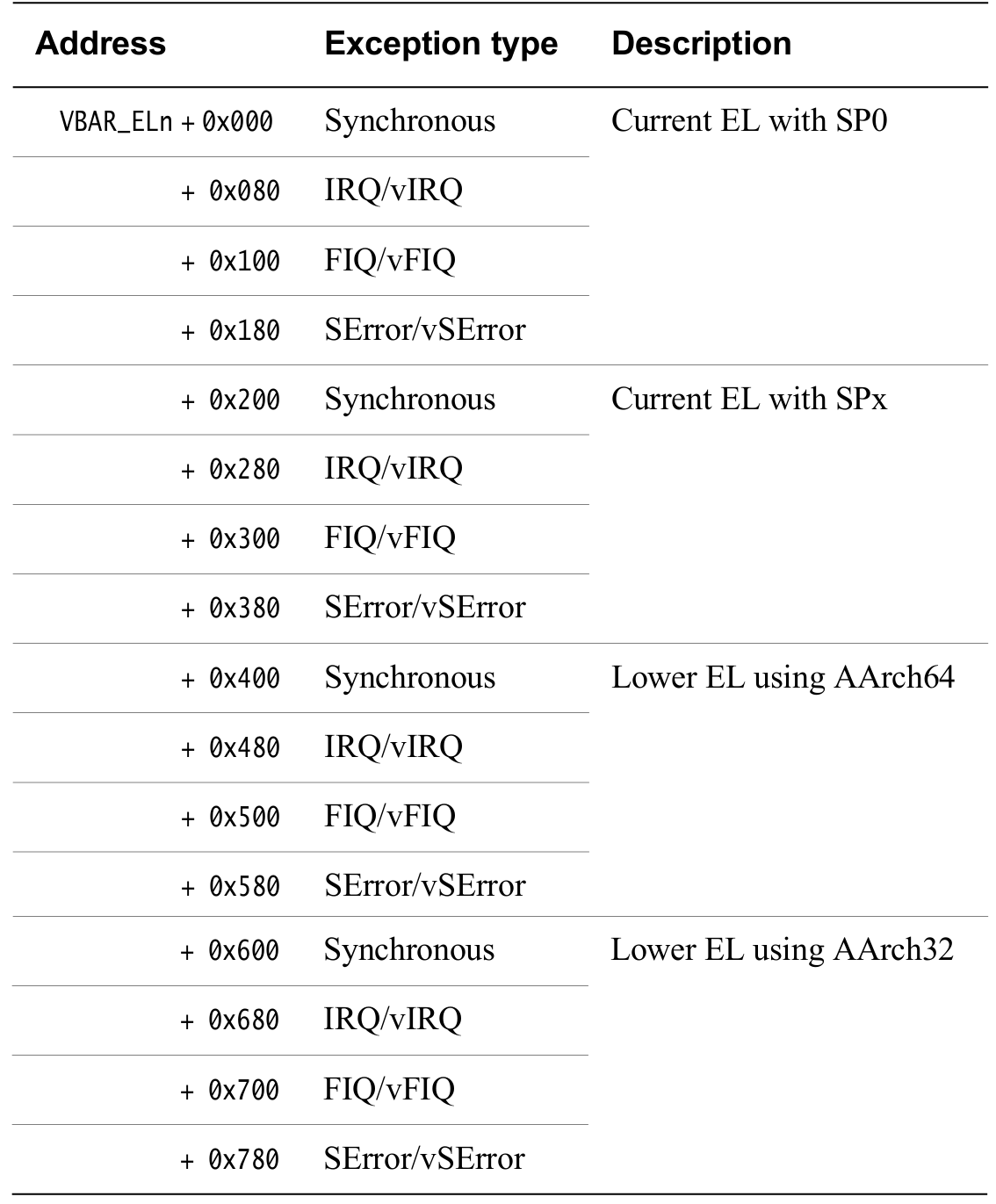
## The basics of Arm TrustZone

Arm TrustZone creates an isolated secure world which can be used to provide confidentiality and integrity to the system. It is used on billions of applications' processors to protect high-value code and data for diverse use cases including authentication, payment, content protection and enterprise. On application processors it is frequently used to provide a security boundary for a GlobalPlatform Trusted Execution Environment.

Historically, the opposite of Privilege is User, and User always corresponds to EL0. This is the level corresponding to EL0. In first ARM core, only the equivalent of EL0 and EL1 were defined. The difference is mostly that EL0 does not have access to system registers (Exceptions, program the MMU, etc.) which are reserved to EL1.

Now the recent cores define 2 additional level which have more privilege, ie more and more system registers available:

* EL2 corresponds to the Hypervisor, which can define virtual machines. A virtual machine generally runs in EL1 and is generally an OS (Linux, Windows, etc.)
* EL3 is the level above which is called Monitor Mode. This extra level is used to run security applications (Trustzone).



Secure Monitor Calls (SMC) generate synchronous exceptions handled by the handler stored at VBAR\_EL3 + 0x600 (Synchronous EL3 exception handler)

## TrustZone commercial implementations

Several commercial implementations have been developed, the three most present on mobile platforms being:

* Kinibi from Trustonic
* QSEE from Qualcomm
* TrustedCore (TEE Operating System on Huawei)

Numerous vulnerabilities were found in different implementations of TrustZone, but several were also found in the Secure Boot of multiple platforms, allowing to obtain code execution in the TrustZone. Some of these vulnerabilities and their impacts are discussed below.

### Vulnerabilities in TrustZone itself

#### Qualcomm

Several vulnerabilities were found by Gal Beniamini including code execution in [Secure World userland](http://bits-please.blogspot.fr/2016/05/qsee-privilege-escalation-vulnerability.html) privilege escalation to gain code execution in Secure World kernel [via SMC handler](http://bits-please.blogspot.fr/2015/08/full-trustzone-exploit-for-msm8974.html) or via [SVC (syscall in ARM) handler](http://bits-please.blogspot.fr/2016/06/trustzone-kernel-privilege-escalation.html) allowing [KeyMaster Keys extraction](http://bits-please.blogspot.fr/2016/06/extracting-qualcomms-keymaster-keys.html) [Linux Kernel hijacking from TrustZone](http://bits-please.blogspot.fr/2016/05/war-of-worlds-hijacking-linux-kernel.html), and [bootloader unlocking](http://bits-please.blogspot.fr/2016/02/unlocking-motorola-bootloader.html)

Two vulnerabilities were found by Azimuth Security including write primitive and arbitrary [code execution in Qualcomm Trustzone](https://www.blackhat.com/docs/us-14/materials/us-14-Rosenberg-Reflections-on-Trusting-TrustZone.pdf)

A [vulnerability](https://pacsec.jp/psj14/PSJ2014_Josh_PacSec2014-v1.pdf) allows to write a zero dword to any address in the TrustZone Kernel. It can be used to disable memory boundary validation on TrustZone memcpy function, crafting an arbitrary write primitive.

As this paper shows, Qualcomm TrustZone suffers from a single point of failure, there are many individual actors in TrustZone, but a single mistake from one of these players impacts all the other components. A single arbitrary memory write vulnerability, especially in secure world kernel, can make the whole security model collapse.

Finally, using signed comparison instead of unsigned comparison leads to [leaking information from Secure World to Normal World](http://theroot.ninja/disclosures/TRUSTNONE_1.0-11282015.pdf)

#### Trustonic

Unlike QSEE, the TrustZone developed by Trustonic and formerly called t-base, then renamed Kinibi, has a micro-kernel which means there is no single point of failure anymore. Although there is less public research on Trustonic's TrustZone, there are some articles detailing the [t-base](https://medium.com/taszksec/unbox-your-phone-part-i-331bbf44c30c) [internals](https://medium.com/taszksec/unbox-your-phone-part-ii-ae66e779b1d6) and [some vulnerabilities found in Secure World userland](https://medium.com/taszksec/unbox-your-phone-part-iii-7436ffaff7c7).

The acquisition of the TEE OS is easier from a Qualcomm TrustZone than Trustonic TrustZone. The [Qualcomm TrustZone can be acquired](http://bits-please.blogspot.fr/2015/08/exploring-qualcomms-trustzone.html) directly from a block under /dev/block, while the [Trustonic TrustZone must be acquired by reversing sboot, for exemple](https://blog.quarkslab.com/reverse-engineering-samsung-s6-sboot-part-i.html).

### Weaknesses in TrustZone hardening

TrustZone hardening is a crucial point in order to obtain good security properties and slow down the reverse engineering and exploitation process. Nonetheless, several classic hardening points, listed below, are not applied:

* Both implementations have many debugging strings present in production.
* There is no ASLR on Kinibi, and only 9 ASLR bits on QSEE.
* There are no Stack Cookies on Kinibi. Conversely there are Stack Cookies on QSEE.

It is interesting to note that some TrustZone implementations, such as the one developed for the Nintendo Switch, are stored encrypted and are decrypted during the boot process, adding an additional difficulty to the attacker.

## Conclusion

On the one hand, the security model implemented with the TrustZone technology provides additional segmentation through the separation of Secure World and Normal World, allowing, for example, DRM solutions to protect against an hostile environment such as an infected system on both user-land and kernel-land.

On the other hand, the development of an entire operating system is a daunting task that often involves many bugs. Operating systems running TrustZone are no exception to the rule. A development error leading to memory corruption in the Secure World kernel causes total system corruption in Secure World, making its security obsolete. It also totally compromises the Normal World which is accessible from the Secure World.

Finally, compromising the TEE OS can be done before it is even executed if a vulnerability is found in the secure boot chain, as [has](https://alephsecurity.com/2018/01/22/qualcomm-edl-1/) [been](https://blog.quarkslab.com/vulnerabilities-in-high-assurance-boot-of-nxp-imx-microprocessors.html) [the](https://www.reddit.com/r/SwitchHacks/comments/7rq0cu/jamais_vu_a_100_trustzone_code_execution_exploit/) [case](https://twitter.com/sciresm/status/964619151913336833?lang=en) [several](https://twitter.com/ktemkin/status/964780654142484481?lang=en) [times](https://twitter.com/fail0verflow/status/953084954145181696?lang=en)