# Data Protection using Trusted Execution Environments

Vats, Mudit
Computer Engineering
*Syracuse University*
Syracuse, NY
mpvats@syr.edu

***Abstract*— In this paper, the topic of Data Protection using Trusted Execution Environments is presented. The benefits of using executing code in hardware enforced isolation are proposed. Challenges and how to overcome them are also discussed.**

**Keywords—TEE, trusted execution engine, ARM, Intel, TrustZone, SGX, Software Guard Extension, Dynamic Application Loader, DAL, encryption, enclave.**

## I. Introduction

The problem we are discussing in this paper is the problem of data protection. The need for data protection exists in many common scenarios. A few use cases are presented below.

### A. *Digital Rights Management (DRM)*

Digitally downloaded data or streamed content requires protection so only authorized users can listen or view the content.

For example, someone may download music they purchased. This downloaded music file may be protected by DRM software which encrypts the media and then uses a key unique to the end user's machine and trusted by the content publisher to allow playing of the media. Anyone who copies the media cannot play the content since they do not have the correct access rights (i.e. the key) to play it.

### B. *Retail Experience*

In the retail world, Point of Sale (POS) machines need to process transactions so that customers can purchase their merchandise. It's important that their credit card information, including name, card number, PIN are protected throughout the transaction.

Further, it is important that the actual transaction is protected so that the banks can securely receive the transaction information without the fear that the contents of the transaction have been manipulated in any way.

In the case of deployment, where POS systems need to be turned on at their destination, the POS may need to attest to a server to prove to the secure code attempting to run on the that the system is valid and should, indeed, be run on the POS. This prevents man in the middle type attack where a malicious version of the code is copied to the system and attempts to run potentially forwarding sensitive data to another web site.

### C. *Secure Code*

In the Secure Code case, a software developer may want to securely run some code in a protected manner such that the code that running and it's outputs are fully encrypted. In this case, the code would need to be encrypted on the platform and keys provisioned to the device so that the code may be securely decrypted and executed in the trusted execution environment.

All of these examples require data to be protected. Whether it's code, keys, media, transactions or any other content, it's clear that allowing this data in the clear without protection could open-up the system to security vulnerabilities that can cause data loss, monetary loss and/or exposure of private data.

## II. Software Solutions

While software solutions exist which allow the operating system to enforce data protection, these solutions can be at risk since they depend on the operating system and vulnerabilities in the operating system can compromise the secrets on the platform. Additionally, on a personal computer (PC) for example, often times users are administrators who have full access to the system. If a rogue user gains access to a system as root they can attempt to defeat whichever protection that may exists on the system and potentially compromise any data protection the operating system provides.

Another concern with software-based data protection solution is that of performance. Any extra security software is typically an added tax on the overall system performance since operating system cycles are needed to devote to tasks such as encryption/decryption, access control and resource partitioning. All of these are very complex activities which, by allowing software to manage these, increate the potential attack surface of the operating system which, once again, can compromise whatever security it's intending to implement.

## III. Hardware

This paper is about hardware based security protections. In particular, Trusted Execution Environments (TEE). A TEE is an secure environment where code can run in an isolated manner such that it's not susceptible to operating system vulnerabilities.

By using hardware based mechanisms, we can separate the operating system from the security hardware thus preventing weaknesses in the operating system to affect the security mechanism. For example, if the OS gets compromised or a malicious user is attempting to compromise the system, the keys, algorithms and data which are protected by hardware based security will not be compromised. *Of course, anything can theoretically be compromised, but hacking hardware is much more complex then hacking software. There are many scholarly articles about this topic. It will not be addressed here.*

This paper focuses on hardware based security mechanisms that are supported by the main CPU which allow code (software) to execute in secure / trusted environments with the sole purpose of data protection. While there are many TEEs, we discuss three interesting technologies: ARM TrustZone, Intel Dynamic Application Loader and Intel Software Guard Extensions

All three of these technologies create a secure trusted execution environment which is sufficiently isolated from the general purpose "rich" operating system.

## IV. Trusted Exectution Environment

Before delving into these technologies, it is importing to discuss the Trusted Execution Environment (TEE). TEE is, as the name suggests, an environment where application can run that is isolated from the host operating system. The idea is that the host operating system is untrusted and should not be able to peer into or access the code running in the TEE. This protection is hardware enforced so that any part of the untrusted operating system and applications cannot access the trusted code or hardware resources. The trusted code running in the TEE should be able to prove that it has not been tampered with (attestation), should be able seal secrets to persistent storage and be able to ensure the confidentiality of the application data.

The figure below shows a common approach to trusted execution environments. We can see two distinct parts. On the left side, we see the Untrusted Area. This area contains a Rich Execution Environment (REE) which is typically a feature-rich operating system. On the right side, we see the Trusted Area with the Trusted Execution Environment. This is the area which contains the secure operating system or code which provides some functionality that needs to be protected. There is a dotted line between the two which show the Hardware Separation between both environments and also to convey that hardware enforcement exists to keep the two worlds separate.

Additionally, we can see that memory is also separated to suggest that the TEE has its own, isolated from the untrusted area, memory which only it can access. Also indicated are peripherals via the system bus which in the figure are separated from the untrusted and trusted Areas and may be shared by both areas, or only accessible by one or the other, or be able to be securely accessed or provide no secure access whatsoever.

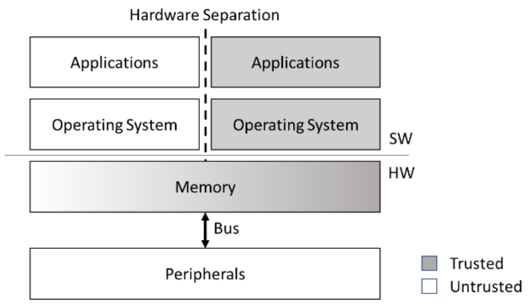


*Figure 1: TEE Architecture*

While not shown in the figure there is usually some facility implemented via software and hardware which allows communication to and from the untrusted and trusted areas.

Typically, the untrusted area, for whichever purpose, calls into the trusted area for secure processing of some data. For example, it may be the case that the device needs to encrypt some data and that the key for that data is only accessible by the trusted area. This may be the case the key was provisioned and initially sealed by the trusted area. In that case, only the trusted area can access the sealed data key and use it to encrypt the untrusted data.

## V. Trust Zone

TrustZone is an implementation of a TEE for the ARM architecture. The figure below shows the architecture.

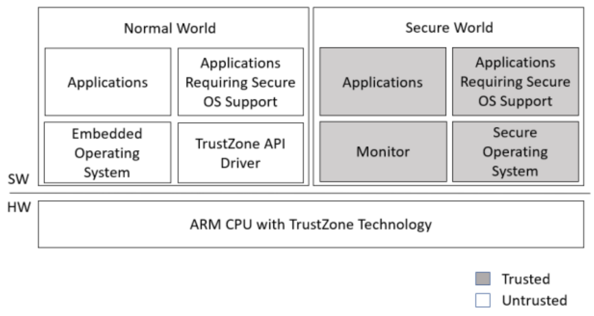

Normal World
Secure World
Applications
Applications Requiring Secure OS Support
Applications
Applications Requiring Secure OS Support
Embedded Operating System
TrustZone API Driver
Monitor
Secure Operating System
SW
HW
ARM CPU with TrustZone Technology
Trusted
Untrusted

*Figure 2: TrustZone Architecture*

TrustZone partitions the system into two worlds. A Normal World and a Secure World. The Normal World is considered the untrusted zone. The Secure World is considered the trusted zone.

### A. *The Normal World*

The Normal World contains a rich operating system (or embedded operating system), applications, applications which require trusted secure OS support and a TrustZone TZ driver. The components that reside above the Embedded Operating System are discussed.

#### 1) *Applications*

Applications which don't require trusted services. These can be user interfaces, middleware, tools/utilities or any other applications needed by the system.

#### 2) *Applications Requiring Secure OS Support*

These are applications which require some secure processing from the Secure World. These can be DRM, encryption/decryption, secure algorithm or any other processing which requires confidentiality. The Normal world can calls into the trusted world via the Secure Monitor Call (SMC).

#### 3) *TrustZone API TZ Driver*

This is the driver which allows applications requiring secure operating system support to communicate with the Secure World. TZAPI is used for Normal World and Secure World to communicate with each other.

### B. *The Secure World*

The Secure World contains a Secure OS Trusted Applications and a Monitor. These components are described below.

#### 1) *Secure OS*

This is a full-blown operating system, or a library OS or an executable binary.

#### 2) *Trusted Application*

These are applications which exist which provide some secure service. These services would contain the actual implementations for the DRM, encryption/decryption and secure processing which is required by the Normal "untrusted" World.

#### 3) *Monitor*

"Monitor Mode" is the processor state which the Secure Zone operates under. In the context of the figure, Monitor is the mechanism the TZAPI communicates with to communicate with the Secure World.

### C. *System Resources*

In ARM's TrustZone, the core is partitioned into two worlds. Additionally, the components which the CPU in each zone interacts with is also partitioned between the two worlds. Some components are part of the same resource, but others are clearly distinct with respect to each zone.

In the figure below, we which components each world can access. Please note that because the Secure World is a TEE, the Normal World cannot access the Secure World's resources.

Normal World has access to several resources. The CPU and Cache are shared or separated depending on ARM architecture. RAM is separate, but part of the same system RAM. Peripheral devices such as Ethernet, USB, MMC, GPU and UART are only available to Normal World.

Secure World has access to its own resources. The CPU and Cache are shared or separated depending on ARM architecture. The RAM is separate, but part of the same system RAM. Exclusive to Secure World are the RTC, GPIO, Graphics, I2C, Security Controller. These are only available to Trusted World.

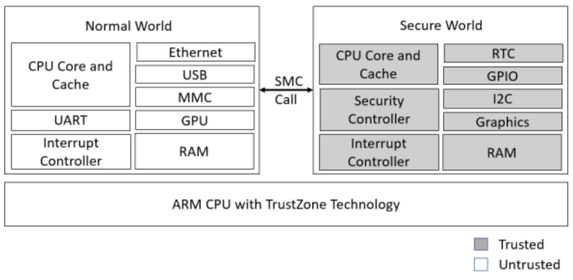


*Figure 3: TrustZone Devices*

Access to components is determined by the Non-Secure (NS) bit. The Secure Configuration Register contains the NS bit which if set, the OS running in the non-secure mode can only access a the normal world resource. If not set, it can access its resource. The secure world's resources are secure, as the name suggests. That means that while the normal world cannot access the secure worlds' resource because the NS bit prevents access from normal mode to access secure mode resources.

### D. *Encryption*

An interesting thing to note is that while access control is implemented to safely keep system resources separate between worlds, this does not imply that memory, bus transactions or any other non-persistent data is encrypted in the Secure World. Access is certainly enforced by the hardware, but there are no protections against physical attacks. For example, an in-target probes can be connected to the platform which can access system details, such as registers, memory or bus transactions. In this situation, secure memory would be unencrypted and in the clear.

### E. *Limitations*

The split architecture only allows two modes only. TrustZone cannot run more than one secure environment. In contrast SGX or Dynamic Application Loader allow many enclaves or applets to be loaded simultaneously. The partitioning and hardware configuration is dependent on the SOC designer.

Additionally, TrustZone does not directly support secure storage; i.e. the ability to seal data. Sealing data is the idea that data which is used in the TEE can be encrypted and saved to disk allowing only the TEE application the ability to access it in the future. By doing this, the data is persisted but protected from a confidentiality perspective.

Finally, ARM TrustZone secures the TEE by access control. This is unlike SGX which encrypts memory pages at rest. As mentioned in the Encryption section, encrypting pages would help mitigate the risk data compromise of physical attacks. Of course, encryption does have its performance considerations and considering ARM devices are typically in smaller embedded, mobile or portable device, perhaps this is an intentional trade-off.

## VI. Dynamic Application Loader (DAL)

The Dynamic Application Loader is an Intel technology which allows applications to run in a TEE. The figure below describes the DAL.

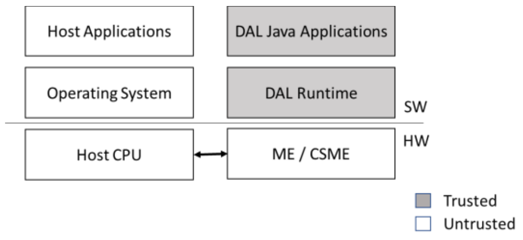


*Figure 4: Dynamic Application Loader*

DAL's TEE executes on a separate processor in the Intel Management Engine which is also known as the Converged Security and Management Engine (CSME). The ME is not the host processor. It's an embedded processor on the System on a Chip (SOC). In current architectures, this processor is a Quark x86 core running Minix OS. Prior to that it was an ARC processor.

Trusted applications are Java applets which run in a trusted JVM running on the ME. Applications have their own processor, firmware and memory, but cannot directly access peripherals. This is in contrast to ARM's TrustZone where the Secure World may access peripherals depending on the configuration and implementation of the platform. The trusted applications uses Java Host Interface (JHI) with the ME driver to allow applications to communicate with the applet service. This allows the host application to communicate with the trusted applet.

Applets are signed by Intel or a third party which via attestation, can verify themselves as valid application that can run on the platform. As previously mentioned, this prevents man in the middle type attacks if the application is modified or replaced. As part of the attestation, the DAL can also use provisioning to inject secrets onto the platform. This allows keys to securely be placed onto the platform after deployment.

Unlike ARM's TrustZone, the DAL can seal data. By sealing, this means that data can be written to persistent storage in an encrypted manner such that only the applet can decrypt it and utilize it.

The DAL supports a very rich out of the box development experience by proving a robust set of APIs which can be used in the DAL Java applet. The SDK includes APIs for Cryptography, Events, Inter-Applet Communications, Monotonic Counters, SSL/TLS, Secure Storage, and facilities for Applet Attestation.

### A. *Limitations*

Since the processor in the ME/CSME is not the host processor, it can be very slow. As previously mentioned, Intel was originally using an ARC processor and recently switched to an x86 based Quark processor. Both processors are relatively slow compared to today's desktop processors and are meant for specialized low-power computing. As such, transaction heavy operation utilizing the ME may not be as performant as using an ARM TrustZone or Intel SGX type solution where the host processor is used for all operations.

## VII. SOFTWARE GUARD EXTENSIONS (SGX)

Software Guard Extensions (SGX) is an implementation of a TEE for the Intel architecture. It's currently available in most desktop processors and some server chips as well. The figure below shows the architecture.

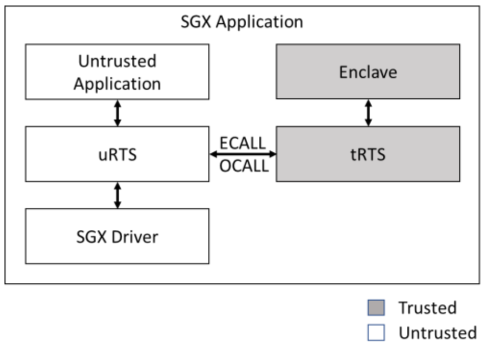


*Figure 5: SGX High-Level*

Each of the components are described below, however, a brief explanation of the application flow is discussed here. An SGX application consists of one or more enclaves, which is the trusted code. Untrusted code, which runs on the host operating system, loads the enclave (i.e. the trusted code), and uses software methods to interact with the enclave. The enclave is protected using hardware mechanisms to ensure enclave data is kept private and inaccessible by the untrusted application. The untrusted application and trusted enclave use run-time services to facilitate calling functionality and data sharing between the untrusted and trusted worlds.

### A. *SGX Archtiecture*

#### 1) *Untrusted Application*

This is the application which loads and starts the trusted enclave. This application is untrusted as the name suggest. It essentially controls the lifetime of the enclave. Additionally, by means of ECALLs it can call into the enclave. Also by exposing OCALLS, it can allow the enclave to call into the untrusted application.

#### 2) *Enclave*

This is the trusted code which implements some trusted functionality, such as, DRM, encryption/decryption etc. This code runs in the TEE. The enclave exposes ECALLs which the untrusted application can call into that make its services available to the normal world. The ECALL implementation exists in the enclave.

#### 3) *uRTS Untrusted Run-Time System*

This code allows ECALLs and OCALLs to and from the enclave. This is the run-time service that runs on the host operating system which facilitates inter host-enclave communication.

#### 4) *tRTS – Trusted Run-Time System*

This code allows ECALLs and OCALLs to and from the enclave. This is the run-time service that runs on the trusted execution environment to help facilitates inter host-enclave communication.

#### 5) *Driver*

This is the SGX driver which support functionality and to access to the SGX device. This may also include additional software components that facilitate the attestation process. Drivers exist for both Windows and Linux operating systems.

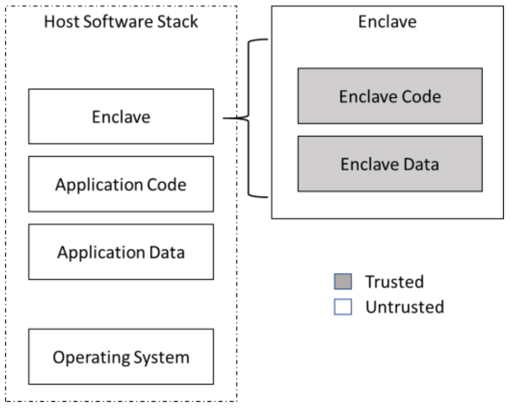


*Figure 6: Loading an Enclave*

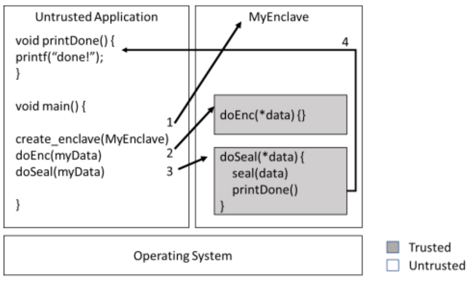
The idea with SGX is that an untrusted application loads the enclave which is a separate binary. This is depicted in the figure as "Enclave". The enclave gets loaded into the untrusted processes memory in the Host Software Stack, but the memory is protected by the CPU hardware so that the untrusted application cannot see the pages where the enclave is loaded. This protection is hardware enforced by the Enclave Page Cache (EPC). Additionally, the pages for the enclave are encrypted using the Memory Encryption Engine (MEE). The encrypted pages stay encrypted in memory and, when brought into the CPU, they are decrypted. By encrypting enclave pages, memory is protected against physical snooping type attacks.

### B. *Programming Model*

The figure below shows the execution of an enclave. In particular, it shows how an untrusted application calls into the enclave and how the enclave calls into the untrusted application. In the figure, we see in step one, the enclave is created. This loads the enclave into the untrusted processes memory space as previously mentioned. In step two, we see that doEnc() is called with some untrusted data. This call is implemented in the enclave, but with help from the run-time services, the untrusted application can call into the enclave. The run-time services copy the myData from untrusted memory to trusted memory and then calls into the enclaves doEnc() function. After the function returns, execution continues in the untrusted application.

In step three, we see the untrusted application calling a doSeal() function which resides in the enclave. The same steps occur as mentioned for step 2. In this case, however, the trusted application (i.e. the enclave) calls into the untrusted application to display completion status. This is step four. Once again, the run-time services help facilitate this communication to make sure the appropriate protections exist in the trusted world.

The main point is that the untrusted and trusted worlds can call into each other but rely on hardware mechanisms to ensure resources are protected along the way.



```
void printDone() {
printf("done!");
}

void main() {

create_enclave(MyEnclave)
doEnc(myData)
doSeal(myData)

}
```

```
doEnc(*data) {}
```

```
doSeal(*data) {
  seal(data)
  printDone()
}
```

*Figure 7: Enclave Execution*

## C. Access Control

Like ARM's TrustZone, there is hardware in place to control access to memory resources for untrusted and trusted applications. But, unlike ARM's TrustZone, memory is protected by use of encryption. By using encryption, if a physical attack occurs and the memory pages are read, they will be encrypted and useless to the attacker.

*Of course, a motivated attacker can attempt to decode the encryption, but due to the current computing speed, encryption algorithm, number of key bits etc. it's very difficult to achieve.*

Like ARM's TrustZone, secure enclaves run on host processor, so they benefit from the full speed of the processor. Additionally, multiple enclaves can run on the same platform. This is in contrast to ARM's TrustZone where only one TEE is able to run.

The EPC memory is limited to 128MB total for all enclaves. The idea is only secure code / critical code should be run in the enclave, so 128 MB may not be an issue for many applications, but there can be issue when trying to run multiple enclaves. Swapping can happen on Linux, but that is very slow and the tax of paging may offset the benefits of using this capability.

On the topic of encryption, SGX can natively support sealing data to disk. This can be accomplished calling into the enclave with the data that needs to be sealed from the untrusted application. The enclave can then seal the data and return it back to the application. Since sealing is called in the enclave, data from the trusted application can be sealed in the enclave as well, but the untrusted application will need to be utilized, via an OCALL, to write the data to disk.

## D. Remote Attestation and Provisioning

"Remote attestation provides verification for three things: the application's identity, its intactness (that it has not been tampered with), and that it is running securely within an enclave on an Intel SGX enabled platform" [16]

SGX can use IAS to provision secrets onto the platform. By using the Intel Attestation Service to prove to an IAS service that the enclave should legitimately be run on this platform, secrets can also be sent to the platform and sealed to disk.

## E. Limitations

Code is not encrypted. This is the case with the other TEEs discussing in this paper as well. This means that the binary which executes can be disassembled to know what the code does. Services do exist for SGX which can be use to encrypt the enclave (ex. Protected Code Loader), but as it is, the SGX focuses data protection and not code protection.

Another limitation, as mentioned, is the EPC size. When considering a Library OS's such as Graphene where an unmodified Linux binary can run in an SGX enclave, 128 MB may not be large enough for applications to work with. While paging may allow larger amounts of memory to be utilized, paging into a small window leads to a lot of page thrashing which can result in performance degradation.

Finally, while memory is protected via the EPC, access to all IO (peripherals) is not. The SGX enclave cannot directly access peripherals, like USB devices. In the future, however, it would be beneficial if SGX dedicated IO buses existed which can allow end-to-end, from peripheral device to enclave, encryption by hardware.

## VIII. Conclusion

In this paper we discussed how data can be protected using hardware based technologies. We started off by presenting use cases and reasons why TEEs are valuable. We then presented a high-level overview on how TEEs work. We followed-up with discsussions of ARM's TrustZone, Intel's Dynamic Application Loader and Intel's Software Guard Extensions.

All three technologies use hardware protection mechanisms to control and ensure untrusted parts can not interfere with trusted parts. In TrustZone, access control is used to identify which world a particular access belongs to. In SGX, sperate page tables and encryption are used to ensure there is clear distinction and protection between memory resources. In the ME/CSME, a separate processor with its own memory is utilized. All provide a mechanism for each world to communicate with each other in a protected manner. Some of the TEE technologies allow multiple trusted worlds, others (TrustZone) allow one. *Which one to use?* It all depends on real-world use cases, business resources and technical needs. The only guarantee is that if multiple domains are needed in a system where at least one domain must do some trusted operation, then there is a need for a hardware-enforced TEE.

The call to action is to use TEEs in your designs to protect your data assets.

## IX. References

[1] M. Zhang, Q. Zhang, S. Zhao, Z. Shi, Y. Guan, “SoftME: A Software Based Memory Protection Approach for TEE System to Resist Physical Attacks”. https://www.hindawi.com/journals/scn/2019/8690853/

[2] H. Vill, “SGX attestation process”. https://courses.cs.ut.ee/MTAT.07.022/2017_spring/uploads/Main/hiie-report-s16-17.pdf

[3] Victor Costan and Srinivas Devadas, “Intel SGX Explained”. https://css.csail.mit.edu/6.858/2017/readings/costan-sgx.pdf

[4] Wikipedia, “Trusted exection environment”, https://en.wikipedia.org/wiki/Trusted_execution_environment

[5] Secure Technology Alliance, “Trusted Execution (TEE) 101: A Primer”. https://www.securetechalliance.org/wp-content/uploads/TEE-101-White-Paper-FINAL2-April-2018.pdf

[6] Global Platform, “Introduction to Trusted Exection Environments”. https://globalplatform.org/wp-content/uploads/2018/05/Introduction-to-Trusted-Execution-Environment-15May2018.pdf

[7] Wikipedia, “Intel Management Engine”. https://en.wikipedia.org/wiki/Intel_Management_Engine

[8] ARM, “The TrustZone API”. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.prd29-genc-009492c/CACCICDE.html

[9] Intel, “Dynamic Application Loader”, https://software.intel.com/en-us/dal/overview

[10] J. Gonzalez, “Operating System Support for Run-Time Security with a Trusted Exection Environment”. https://www.researchgate.net/figure/Trusted-Execution-Environment-TEE-definition-In-order-to-support-a-TEE-a-device_fig1_297732884

[11] A. Adamski, “Overview of Intel SGX - Part 1 SGX Internals & 2 SGX Externals”. https://blog.quarkslab.com/tag/intel-sgx.html

[12] J. Guilbon “Introduction to Trusted Execution Environment: ARM’s TrustZone”. https://blog.quarkslab.com/introduction-to-trusted-execution-environment-arms-trustzone.html

[13] B. Ngabonziza, D. Martin, A. Bailey, H. Cho, S. Martin, “TrustZone Explained: Architectural Features and Use Cases”. http://sefcom.asu.edu/publications/trustzone-explained-cic2016.pdf

[14] S. Mofrad, F. Zhang, S. Lu, W. Shi, “A Comparison Study of Intel SGX and AMD Memory Encryption Technology”. http://www.cs.wayne.edu/fengwei/paper/sgxsev-hasp18.pdf

[15] N. Zeldovich, H. Kannan, M. Dalton, C. Kozyrakis, “Hardware Enforcement of Applicatoin Security Policies Using Tagged Memory”. http://www.cs.wayne.edu/fengwei/paper/sgxsev-hasp18.pdf

[16] SSLab, Georgia Tech “Remote Attestation”. https://gts3.org/pages/remote-attestation.html

[17] Intel, “Dynamic Applicatoin Loader Developers Guide”. https://software.intel.com/en-us/dal-developer-guide-architecture-applets