Omnishield – An Overview & Requirements

a White Paper

Filename	:	Ominishield_WhitePaper_IMG_2016.docx
Version	:	1.0
Issue Date	:	03 Nov 2016
Author	:	Imagination Technologies Limited
Number	:	MD01185

Contents

List of Figures

1. Omnishield

1.1. Overview – Security by Separation

Omnishield provides security through separation of memory spaces used by each system function. Each of these separate memory spaces is known as a “Domain”. Each system function can only use its own domain and normally can’t access the domains used by other functions.

A competitive advantage of Omnishield versus the ARM Trustzone technology is that multiple domains can be created as required by each system. By having multiple domains, single functions can be isolated from the rest of the system. This allows for the advantageous situation where mutually-distrusting applications not having to trust each other. In Trustzone, mutually distrusting applications do have to trust each other as they would all reside in a single Trusted World. The architecture allows up to 255 domains, particular implementations might only allow lower number of domains.

Access to memory spaces which are shared among multiple domains is programmatically controlled by privileged software within the system.

1.2. Virtualization and Domains

The different memory domains are created by using a second MMU within the CPU. This second MMU is a key feature of CPUs following the MIPS Virtualization architecture.

By using this second mapping, the actual physical address range used by a function will be different from an identical copy of that function even though both copies are using the same virtual addresses.

1.3. Scope of protection and attack types

Omnishield is targeted at protecting the system from malicious software attacks which is trying to either modify the system behaviour or steal assets/secrets from the system. Omnishield is also useful for software bugs which might cause an operating system to crash or otherwise malfunction.

Omnishield is not targeted at physical attacks such as logic analyser probes placed on board traces. There are other techniques which deal with physical attacks. Those other techniques are not discussed in this document.

Omnishield is not targeted at side-channel attacks such as differential power analysis. There are other techniques which deal with such side-channel attacks. Those other techniques are not discussed in this document.

1.4. Hypervisor

The Hypervisor is a Software component which configures and manages these memory domains. Besides the domain management, the hypervisor is also responsible for loading the software which would be running in those domains.

The software within a domain is called a ‘Guest”. A Guest could be an entire Operating system or as simple as a stand-alone application.

The hypervisor is also responsible for context switching among these Guests as needed by the system as well as dealing with events that affect all Guests.

Omnishield requires the usage of a Type-I hypervisor. This type is also known as bare-iron hypervisor or native hypervisor.

Omnishield also requires that the hypervisor support full virtualization of the CPU. That is, the hypervisor uses the features of the MIPS Virtualization architecture – the 2nd-level MMU, the Guest COP0 context, the interrupt virtualization support.

1.5. Root of Trust and Trusted Elements

The Root of Trust (Rot) is an entity that is responsible to authenticate the software that is running on the system. It is called the Root of Trust as it is designed so that it executes as designed from the factory without the possibility of tampering. The Root of Trust must be the first active component within the system upon system power-up to ensure that no tampering is done with the rest of the system. (Throughout this document, we will use the acronym "**RoT**" for "Root of Trust".) For this authentication functionality, the RoT normally supplies encryption, decryption and hashing functions to the rest of the system.

In very simple systems, some code held in un-mutable memory (ROM or OTP programmed at the factory) may function as the RoT.

More sophisticated systems would have a separate subsystem that acts as the RoT. This separate subsystem will be known in this document as the "Trusted Element". We will use the acronym "**TE**" for "Trusted Element" in this document.

- The RoT could be implemented as a separate Hardware block. This type is known as a Hareware Trusted Element (TE).
- The RoT could be implemented as a separate execution mode of the main-line processor. This type is known as a Virtual Trusted Element. One way to implement this is using one of the Guest OSes within the MIPS Virtualization architecture.

One common method of protecting the Trusted Element from the rest of the system is one-way communication paths. The rest of the system can only send requests into the TE. The TE only sends back results or responses to such commands.

The internal workings of the TE are not visible to the rest of the system. For a HW TE, this is achieved through physically separate resources that are dedicated to the TE. For a Virtual TE, this is achieved through memory space separation.

1.6. Secure Boot and the Chain of Trust

As previously described, it's the responsibility of the RoT to ensure the authenticity and integrity of the software that is running on the system. This is accomplished through a chain-of-trust that is built during the Secure Boot sequence.

What follows is an example secure boot sequence.

The required boot sequence is a multi-phase affair. Starting with a very simple level-0 boot loader whose only function is to authenticate the next boot-loader; load it into executable memory and then transfer control to that next boot-loader. This level-0 boot loader must be very simple as it's held in un-mutable memory from the factory. It can't be changed after the device is initially manufactured. Because this code cannot be changed or modified by the user, the code can be trusted to execute as programmed from the factory.

What is described as "authentication" usually includes these steps:

- Confirming the correctness of a cryptographic hash function of the payload. This is to confirm that the payload content has not been tampered with and is as intended from the factory.
- Decrypting the payload content after the hash has been confirmed.

Let's call the next boot-loader the level-1 boot loader. The level-1 boot-loader's responsibility is to make available the encryption/decryption/hashing functions and set up any related hardware (if available). This is allows subsequent phases to use more cryptographically strong techniques for authentication. The level-1 boot-loader also needs to authenticate the next boot-loader; load it into executable memory and then transfer control to that next boot-loader. Because the level-1 boot loader was previously authenticated by the level-0 boot-loader, we know that it hasn't been tampered with and thus can be trusted to execute as programmed from the factory. This is the first link of the chain of trust.

Let's call the next boot-loader the level-2 boot loader. The level-2 boot-loader's responsibility is to authenticate the hypervisor image; load it into executable memory and then transfer control to the unpacked hypervisor. Similar to the previous phase, the level-2 code was authenticated by the previous phase – this is the 2nd link of the chain of trust.

When the hypervisor executes, its responsibility is to do initial configuration of the system; set up the memory domains; authenticate its Guests; load the Guests into executable memory and then transfer control to the first Guest it wants to run. The hypervisor was authenticated by the previous boot loader, thus can be trusted. Because the hypervisor authenticates the Guest code, the Guest can be trusted to execute as programmed from the factory. This is another link of the chain of trust.

One type of attack is software rollback. This is discussed in the system update section. Before the transfer of control is given to the loaded software, the version number of the loaded software should be checked against the version number of the previous update to ensure that an older version of the software is not executed.

1.7. Private Memories for the Root of Trust

To enable the RoT entity to execute its authentication duties, it needs these facilities:

- ROM or OTP memory for the level-0 boot-loader – must be only accessible to the RoT entity if the ROM/OTP code has functionality such as cryptography.
- Persistent storage to hold secrets such as keys – must be only accessible to the RoT entity
- If there are permanent, non-revocable keys burned into the device at the factory, then those keys are must be located in a memory that is persistent, immutable and only accessible to the RoT.
- Execution memory if the RoT entity has a SW component - must be only accessible to the RoT entity
- Data memory for temporary data storage – must be only accessible to the RoT entity

1.8. Other Transaction Masters

So far, we have only discussed the CPU and functionality implemented on the CPU is protected from each other. For other transaction masters, there are two cases:

- Transaction masters which also have Virtualization support – meaning those that have a similar secondary MMU which allows separation of their transactions into the different memory domain.
- Transaction masters which do not have Virtualization support.

1.9. Transaction Masters with Virtualization Support

Examples of this type include Imaginations' PowerVR Series 6-8 GPUs with Virtualization Support.

Series 6 & 7 rely on a customer supplied 2nd-level MMU while Series 8 includes the 2nd-level MMU within the GPU IP product.

These GPUs also provide per-Domain "Kick" registers which is used to wake-up the GPU for new commands. In this way, it appears to each Domain that it has sole access to the GPU.

1.10. Transaction Masters without Virtualization Support

To deal with this type of master, there are a couple of possibilities:

1. The use of an IOMMU to translate the addresses to the appropriate domain. The IOMMU is equivalent to the 2nd-level MMU within the CPU.
2. The use of firewall techniques at the SOC fabric. Firewall is our terminology for logic at either the initiator port of the SOC fabric or the responder port of the SOC fabric which decides if the transaction is allowed. This logic can be programmed so that only certain combinations of initiator and responders are allowed. In this way, the attached IO devices can be protected from certain transaction masters.

1.11. Necessary SOC infrastructure

In the case of using fabric firewall techniques, the SOC fabric needs to have signals which identify the requesting initiator (meaning the MIPS CPU GuestID signals or PowerVR GPU OSID signals - which represent the Domain which is executing).

In the case of the IOMMUs, the IOMMU also needs to receive the signals which identify the requesting initiator. For the IOMMU case, it's optional whether its needed to send those same signals to the fabric.

In the case where there is no entity which remaps the initiator identifiers (equivalent of not using an IOMMU), the CPU GuestIDs and GPU OSIDs should be programmed on both masters to use the same number to represent the same Domain. This is to allow both CPU and GPU to reach the same intended domain. For example, if the system is running 2 Guests (one linux and one Nucleus RTOS), both the CPU and GPU should use the same number N to represent linux and the same number M to represent Nucleus.

In contrast, if the transactions go through an entity which can remap the initiator identifiers (for example an IOMMU can do this number re-mapping), then it's not required for the CPU GuestID to use the same numbers as the GPU OSID to represent a particular domain.

The additional of system security is modularized at the boundaries of the SOC fabric. Either using secondary MMUs, IOMMUs or firewall techniques. This has the benefit of not having to modify the basic structure of the SOC fabric.

1.12. Rich OSes and Trusted Execution Environments

The hypervisor can instantiate two types of Guests:

- Rich OS – this is the term used for a Domain where the end-user can modify the content. Since the end-user can add software that is potentially malicious, this domain is not trusted.
- Trusted Execution Environment (TEE) – This is a Domain which only has content from the factory and can't be modified unless allowed by the appropriate authority. Also, this domain is protected from any Rich OS which is resident in the system. In this document, we will use the acronym "**TEE**" for Trusted Execution Environment.

Very often, within TEEs are small RTOSes which are used to field incoming requests.

TEEs are used to provide services that must be protected from malicious users. Some examples include Conditional Access and Digital-Rights-Management functions for systems dealing with Media (such as movies and music).

One common method of protecting the TEE from the rest of the system is one-way communication paths. The rest of the system can only send requests into the TEE. The TEE only sends back results or responds to such commands.

The internal workings of the TEE are not visible to the rest of the system. This is achieved through memory space separation.

1.13. Key Ladder - Recommendation for Device Key Usage

One of the basic keys to be generated is the key that uniquely identifies a particular copy of the device. This is equivalent to the serial number for that particular copy.

It is strongly recommended to not use this key for any direct run-time function – such as enabling access rights or for encrypted communication with the rest of the world.

Rather, this device-specific key should only be used to create subsequent generations of keys. It is better to use these later generations of keys for the run-time functions. This multiple generations of keys is known as the key ladder.

By using a ladder of keys, this allows keys to be revoked and replaced if some functionality of the system has been compromised. The compromised key can be replaced by generating a new replacement using the earlier generation key as a seed.

If the unique device key was used for any run-time function, it would increase the chances of the key being compromised. If the unique device key was compromised, then the device becomes useless as no replacement key could be generated.

For such reasons, the device unique key must be kept secret from the rest of the world as much as possible.

1.14. Key Management During Manufacture and Production

This is a very large topic, which is only briefly mentioned here as a reminder of its importance.

- Some keys will have to be installed on the device during manufacture (implemented as blowing fuses or programming OTP memory). Such processing steps must be done in a secure facility.
- Records must be kept of the installed keys to keep track of the devices. Such records must be kept in secure computer systems.

1.15. Remote Management of Devices

For connected devices, the problem of how to manage devices remotely brings several issues:

- **Attestation** – How does the managing entity know that it is communicating with the genuine device instead of communicating with an imposter? An imposter might be attempting to steal secrets from the managing entity. Attestation services use cryptographic methods to make certain that the communicating device is genuine.
- **Provisioning** – The managing entity might have to enable certain features or download additional code to the device, in order to configure for its intended usage.
- **Update** – The managing entity might need to update the software on the remote device.
- One type of security attack is rollback. This is where the malicious user attempts to install an older version of code with known flaws. The intent is to exploit those flaws after the older version of software is installed on the device. To prevent this, the update system ought to save an encrypted version number of the successfully installed software. The update system would check this encrypted version number and confirm any install is of a software version which is newer than the already installed version.
- **Diagnostics & status** – The managing entity might need to check status of the remote device.

1.16. Securing Debug

One system facility that needs to be dealt with for security reasons is Debug, especially In-Circuit Hardware Debug. Such Hardware Debug facilities normally give full access to all resources of the system. If such HW debug is accessible to a malicious user, then the malicious user can gain visibility to all operations done by the trusted code-base and thus can gain access to any and all assets of the system. For the MIPS architecture, EJTAG is the name of this type of In-Circuit Hardware Debug.

To prevent such attacks, Hardware debug needs to be either permanently disabled or only enabled when authorized by the factory. Such authorization needs to be gained only through cryptographically strong methods. Otherwise debug authority could be gained through brute-force attacks on simple debug passwords.

2. Requirements of an Omnishield compliant device

OmniS_Requirement1: The device must have a Root-of-Trust entity. It must not be possible to modify this entity outside of the factory. This includes both Hardware and Software components of the Root-of-Trust.

OmniS_Requirement2: The Root-of-Trust entity must implement a cryptographically-strong secure boot mechanism.

OmniS_Requirement3: The first piece of software that is executed after system power-up must be the initial boot-loader. It must not be possible to modify this boot-loader outside of the factory.

OmniS_Requirement4: The following memories must be provided for the RoT entity:

- Immutable, persistent storage for the initial boot-loader
- Persistent storage for system secrets such as keys. Must be only accessible to the RoT entity.
- If there are permanent, non-revokable keys that are placed into the device at the factory, then those keys must be located in a memory which is immutable, persistent and only accessible to the RoT entity.
- Execution memory for the RoT entity if it includes SW components. Must be only accessible to the RoT entity.
- Data memory for the RoT entity. . Must be only accessible to the RoT entity.

OmniS_Requirement5: The device must use a CPU which adheres to the MIPS Virtualization (MIPS Virtualization) architecture. This CPU must implement the 2nd-level MMU with full address mapping capabilities.

OmniS_Requirement6: This MIPS-Virtualization enabled CPU must run a hypervisor to manage the multiple domains and Guest operating systems/applications. The hypervisor must be Type-I and implement full CPU virtualization by using the MIPS Virtualization features.

OmniS_RecommendationA: Other transaction masters within the device ought to use a 2nd-level MMU to direct their transactions to the appropriate physical address locations. This might be a 2nd-level MMU that is specific for that transaction master or a more generic IOMMU.

OmniS_Requirement7: All transaction masters must have a mechanism to direct their transactions to the appropriate physical address locations as used by the security domains. This might require the propagation of the domain numbers (MIPS GuestID or PowerVR OSID) throughout the chip interconnect and I/O devices.

OmniS_Requirement8: All I/O resources must have a mechanism by which they can reject a transaction request from a requestor without appropriate access rights. The set of allowed and not-allowed requesters is defined by the domain address ranges or other mechanisms such as interconnect firewalls.

OmniS_RecommendationB: The device should use a key ladder for the usage of security keys. The keys tied to the hardware instantiation would be reserved to only be used for generation of subsequent generations of keys.

OmniS_Requirement9: Hardware In-circuit debug features (such as MIPS EJTAG) must be disabled during normal operation. Such debug features can only be enabled when authorization has been given from the factory. It must not be possible to enable such debug features without the proper authorization.

3. Components of Omnishield

Omnishield requires a wide range of IP components some mandatory and others optional. It is unrealistic for Imagination Technologies to develop all of these capabilities in house so a number of partnerships are being contemplated. Below is an outline of the major IP components, the approach being taken to secure them and the current status of the work.

Component	Route to develop or acquired	Status
Virtualized CPUs	Internal development	M51xx, I6400, P5600, P6600 available, others under development
Trusted Element (including secure boot, crypto and authentication services)	3rd Party	Imagination working with a 3rd-party TE vendor.
Physically Unclonable Function (PUF)	3rd Party	Imagination working with multiple PUF vendors.
SoC infrastructure with virtualization support	3rd Party	Partnership agreement signed – non-exclusive
Other bus masters without virtualization support	Internal development	IOMMU under development (out of IMG Works) Interconnect firewalls
Hypervisors supporting MIPS virtualization	Partnershipand limited internal development	Various hypervisors available • KernKonzept L4re • PUCRS Hellfire • Seltech Fexerox • MEOS-VZ
Trusted Execution Environment as Guest OS and firmware	Partnership with funding and internal development	3rd party (delivery by CQ4 16). Internal development underway planned to complete in CQ2 2017
Remote Management • **Device attestation services** • **Device provisioning services** • **Device update services** • **Device – status probing & Diagnostic services**	Partnerships via prpl working group	Working on developing a demo of this capability with partners such as Intercede