Qualcomm Technologies, Inc.

An Introduction to Access Control on Qualcomm® Snapdragon™ Platforms

Qing Li, David Hartley and Brian Rosenberg

September 16, 2020

Disclaimer

Qualcomm Technologies, Inc.

Qualcomm Legal Technologies, Inc.

5775 Morehouse Drive

San Diego, CA 92121

U.S.A.

Contents

Introduction

Smartphones are one of the fastest growing technologies of the past decade. Their functionality has expanded beyond making phone calls to encompass a range of rich applications such as HD streaming, multiple cameras, mobile payment, social networking, and location services. At the center of a smartphone design are one or more Systems On a Chip (SoCs), which provide essential hardware support to enable these rich applications to execute with high performance and long battery life.

Many smartphone SOCs have been used as the basis for development of SOCs that are used in adjacent industries such as IOT, Automotive and XR. The need for rich applications and high performance are crucial in these industries as well. Management of credentials, data, service configuration mirrors that of the mobile phone. In this white paper, we talk about smartphones, but the same principles apply to processors based on Snapdragon in adjacent industries as well.

Smartphones handle assets important to different parties such as user private data, account credentials, service configuration, and paid content. They also interact with the outside world through multiple interfaces such as cellular radio, Wi-Fi, and storage cards. To limit the impact if one asset or interface is compromised, SoC components – including software images – are organized into security domains, which should be isolated from each other.

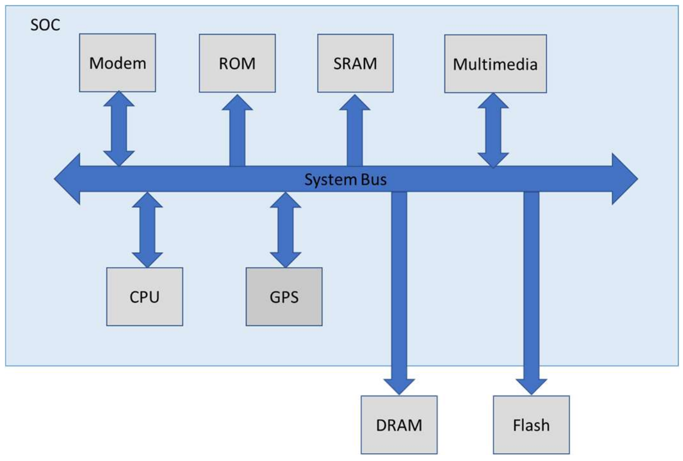

Figure 1: Simplified SoC Block Diagram

The modern design of a mobile SoC is comprised of various components with different roles in SoC and device functionality. Each chip maker chooses the number and types of components to be included in their individual chip design. Figure 1 shows a simplified SoC block diagram including CPU, modem, multimedia, location (GPS), and memory components. The CPU runs an operating system (OS), such as Android or iOS, and provides rich applications to the end users. The modem enables cellular network and other connectivity. In addition to these various computing elements, this example SoC has on-chip ROM and SRAM memories and is paired with an off-chip DRAM memory, and off-chip non-volatile flash memory to store code and data. These components communicate with each other through a system bus.

Each component can contain bus initiators or targets. Bus initiators issue transactions across the system bus towards bus targets. Bus targets are system resources that receive and respond to transactions from the system bus. A transaction is typically a request to read or write a system resource such as a memory location or a hardware register, either within the SoC or in an external component.

SoC access control hardware allows bus initiators and target system resources to be assigned to security domains, with limited cross-domain access as needed to perform their function. Access control (AC) operates by either allowing or blocking transactions issued by bus initiators towards target system resources based on access control policies that are specified using security attributes that accompany each transaction.

Trust Hierarchy on Snapdragon SoCs

A Root of Trust (ROT) domain is a domain that manages its own AC policies as well as the AC policies for subordinate domains. Snapdragon SoCs support multiple ROT domains, which can be mutually distrusting. Different Snapdragon SoCs have different trust architectures. An example trust hierarchy is shown in Figure 2. There are two mutually distrusting ROT domains in this example: ARM TrustZone [1] and our distinct ROT domain. The ARM Hypervisor environment [2] exists under the TrustZone ROT domain, and the trust model between TrustZone and Hypervisor follows ARM design. In this example, the Hypervisor manages three security domains, while our ROT domain does not manage any other security domains.

Security domains with higher privilege have full access to the resources owned by lower-privileged domains. It is possible to protect a lower-privileged domain from the higher-privileged domains using security enclaves or customized logic based on use case requirements. However, this type of protection is outside the scope of this document.

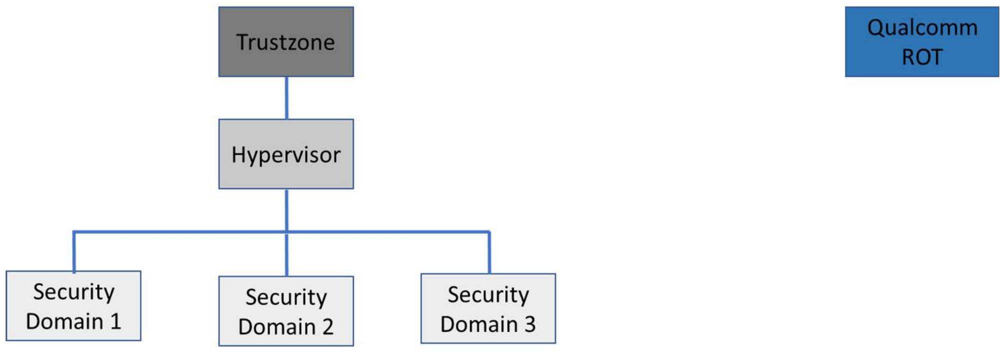

Figure 2 An Example of Trust Hierarchy on Qualcomm SoCs

Access Control Hardware Components

In Snapdragon SoCs, three components are used to provide access control: Virtual Master ID Mapping Table (VMIDMT), External Protection Unit (XPU), and System Memory Management Unit (SMMU). VMIDMT and XPU work together: VMIDMT applies security attributes corresponding to a security domain to transactions (e.g. read/write), while XPU enforces access control policies based on security domains. SMMU maps transactions to security domains and enforces corresponding access control policies. We will further explain the role of each component in this section.

VMIDMT

VMIDMT operates on the initiator-side and provides security attributes corresponding to a security domain for the transactions issued by initiators towards the bus. A single VMIDMT can be shared by more than one initiator. Figure 3 shows an example where two initiators share the same VMIDMT. Each initiator generates identifier attributes that are conveyed to the input ports of the VMIDMT. The identifier attributes are typically fixed in the hardware. VMIDMT transforms the incoming identifier attributes into outgoing security attributes. VMIDMT supports the generation of ROT security attributes such as the NS signal described in ARM TrustZone [1] and QTI-defined security signals. VMIDMT also generates a VMID (Virtual Master ID) security attribute, which is used to further differentiate security sub-domains under the same ROT domain. All of these attributes are generated according to the instantiated hardware parameters and configurations programmed by each ROT domain. The security attribute outputs from VMIDMT are checked by the access control components downstream to determine whether or not a transaction is allowed.

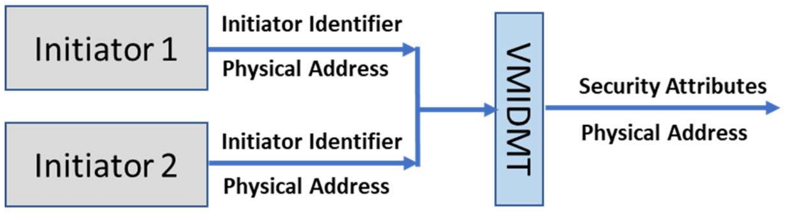

Figure 3: VMIDMT Example

The VMIDMT supports mutually distrusting ROT domains, so each ROT domain can control the generation of security attributes for its own domain. The generation of security attributes for each ROT domain is solely controlled by the ROT domain itself; no ROT domain can influence the generation of security attributes corresponding to any other ROT domains.

XPU

An XPU grants or denies access to target resources by an initiator or a group of initiators based on security attributes from the initiators, XPU hardware instantiation parameters, and software policies. XPU access control policies are defined at the granularity of a set of target addresses, called a resource group, with each XPU protecting many resource groups. The target resources within a resource group can be contiguous or fragmented in the address space depending on the hardware parameters of the XPU. Each resource group is owned by a single security domain and only the owning security domain can configure the access control policies for this resource group. The access control policies determine which other security domains are granted read and/or write access to the resource group. Figure 4 shows an example of XPU protecting two resources.

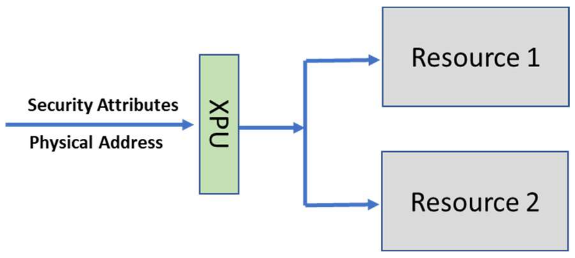

Figure 4: XPU Example

XPU supports three modes of operation: Memory Protection Unit (MPU), Register Protection Unit (RPU), or Address Protection Unit (APU). XPU operating modes are determined at hardware design time, and each individual XPU operates in only one mode.

- In MPU mode, each resource group is a software programmable contiguous address range, typically populated with memory. The start and end addresses need to be properly aligned, in most cases on a 4 KB boundary.
- In RPU mode, each resource group is a fixed contiguous address range of the same size, typically populated with memory mapped registers.

- In APU mode, each resource group can contain fixed fragmented or contiguous address ranges with varying sizes.

The address ranges in a resource group are determined at hardware design time for RPU and APU modes. Each XPU has a fixed number of resource groups determined at hardware design time. The number of resource groups constrains how many distinct access control polices can be supported for each XPU instance.

XPUs also support mutually distrusting ROT domains, so each ROT domain can take ownership of an individual resource group and program the desired access control policies independently of other ROT domains and resource groups. The owning ROT domain can share resources with another by granting the appropriate permissions.

SMMU

The SMMU is a hardware component that performs address translation and access control for bus initiators outside of the CPU. The detailed design of an SMMU can be found in the ARM specifications [3]. This document provides only a high-level introduction to how SMMUs are used in access control on Snapdragon SoCs.

An SMMU can perform two stages of address translation. Stage 1, usually controlled by the CPU OS, maps the virtual addresses visible to applications and the OS kernel to intermediate physical addresses visible to a virtual machine. Stage 2 maps intermediate physical addresses to physical addresses. At each stage, the address mappings – encoded in SW-managed *page tables* – can leave some target address ranges unmapped and restrict the transaction types permitted on others, defining the access control policy to physical address space. This allows a kernel driver in a virtual machine, for example, to allocate a memory buffer, provide a DMA bus initiator with its virtual address range, and rely on the SMMU to translate DMA transactions to the correct physical address ranges for the virtual machine. An SMMU can also ensure that SW does not use DMA bus initiators to circumvent the access control policy imposed on it. Transactions targeting an address not mapped or permitted in the page tables will trigger a bus fault.

Because multiple bus initiators can forward transactions to the same SMMU concurrently, the SMMU supports multiple context banks in parallel, each with its own page tables. An SMMU directs incoming transactions to the correct context bank using signals generated by each initiator, including a stream ID and security state determination (SSD) value. Trusted software configures the SMMU to map these transaction values to a context bank and, hence, to the corresponding page tables. Simple initiators have a single hard-wired stream ID and SSD value, and can be mapped to only one context bank at any time. More complex initiators may have internal logical channels that generate distinct stream ID and SSD values, so that each channel can be mapped to a different SMMU context bank and operate in a different security domain.

SMMU context banks may be configured as secure or not secure. Only TrustZone can map SSD values to secure context banks, and only page tables associated with secure context banks can map virtual addresses to secure physical addresses. In Snapdragon SoCs, the Hypervisor controls both the mapping from non-secure stream IDs to context banks and stage 2 context bank configurations. The CPU OS is permitted to configure only stage 1 context banks for its own use cases. The CPU OS is not able to change context bank mappings or modify stage 2 page tables to access resources not granted by Hypervisor. Similarly, the Hypervisor is not able to re-configure SMMU context banks or modify secure page tables to access TrustZone-owned resources.

Access Control Topology

Access control topology refers to the types and inter-connection of access control components. The choice of initiator-side (permission checked before a transaction is issued to the system bus) or target-side (permission checked closer to the resource being requested) can have important implications for SoC area, power, and performance as well as security.

Target-side Access Control

In target-side access control, a VMIDMT is used to generate security attributes and permission checking is conducted by an XPU. SMMU is not used in this topology. Each request from an initiator goes through a VMIDMT which generates security attributes on the bus. The security attributes are propagated to the target resources and the XPU in front of the resources grants or denies the transaction based on the security attributes from the initiator, destination address, request properties, and XPU policies. A single VMIDMT supports the generation of security attributes for multiple initiators. A target resource can have a dedicated XPU or several target resources can share the same XPU. A target resource that requires no protection does not need to be put behind an XPU. Similarly, if an initiator does not need to access any protected resources, a VMIDMT is not required in the path from the initiator to any target resources. A transaction from this initiator to any protected target resource will be rejected.

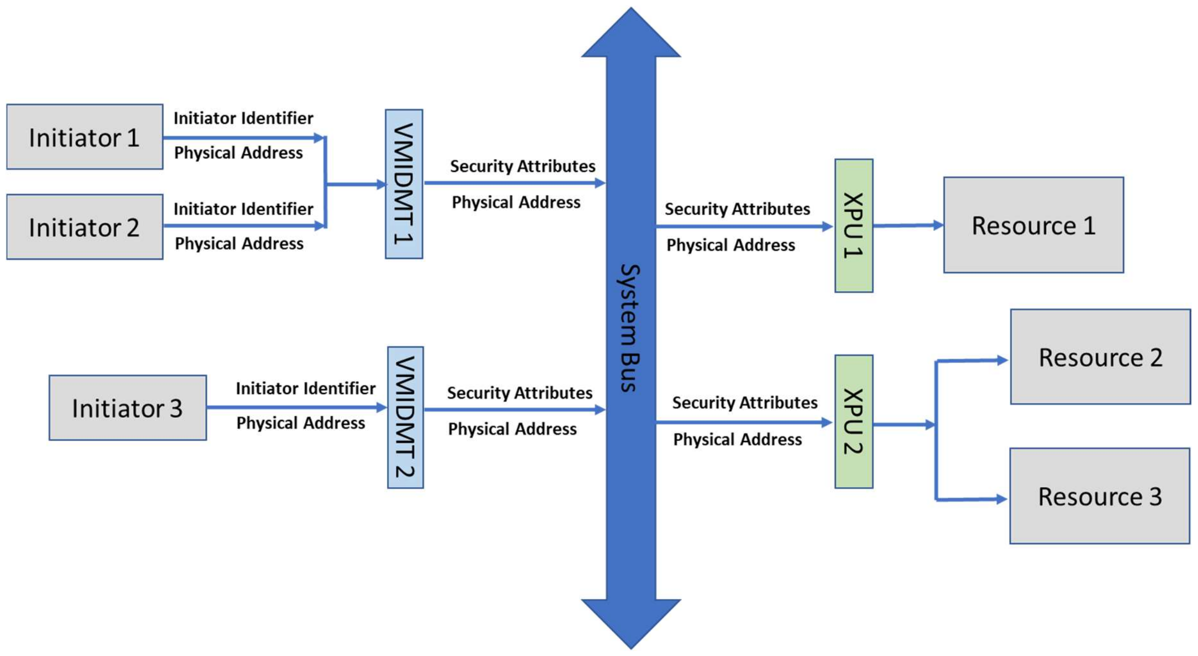

Figure 5: Target-side Access Control

Figure 5 shows an example of target-side access control topology. There are three initiators and three target resources in this example. Initiators 1 and 2 connect to VMIDMT 1, while Initiator 3 is the only initiator connected to VMIDMT 2. On the target side, Resource 1 has a dedicated XPU, and Resources 2 and 3 share the same XPU. When Initiator 1 issues a transaction to access Resource 3, security attributes for Initiator 1 will be generated by VMIDMT 1, and these signals will propagate to the system bus. The system bus decodes the destination address and routes the transaction to XPU 2. XPU 2 checks the security attributes and relevant permissions and grants or denies the transaction. VMIDMT 2 and XPU 1 are not in the path from Initiator 1 to Resource 3 so they are not involved in this transaction.

Initiator-side Access Control

In initiator-side access control, permissions are checked close to the issuing initiator. SMMUs can provide sufficient access control if all SoC components are ARM compliant and every initiator has an SMMU in front of it. However, Snapdragon SoCs include proprietary security attributes which are not supported by SMMU. Further, there are cases where transactions between some initiator-target pairs do not go through an SMMU due to various design considerations, such as performance impact or area cost. To overcome these challenges, the initiator-side access control design makes use of all three access control components, i.e. XPU, VMIDMT and SMMU. As a result, permissions for some resources may be checked close to the target even in a primarily initiator-side design.

Every transaction from an initiator to a protected target resource goes through at least one access control component which performs permission checking. It could be an SMMU or MPU close to the initiator, or an XPU close to the target. When an MPU is used to perform permission checking close to the initiator, we call it an Initiator-side MPU (IS-MPU). The protected target address range for the IS-MPU is the entire range of SoC addresses that can be generated by the initiator. Permissions for a single transaction may also be checked by more than one access control component along the path from an initiator to a target.

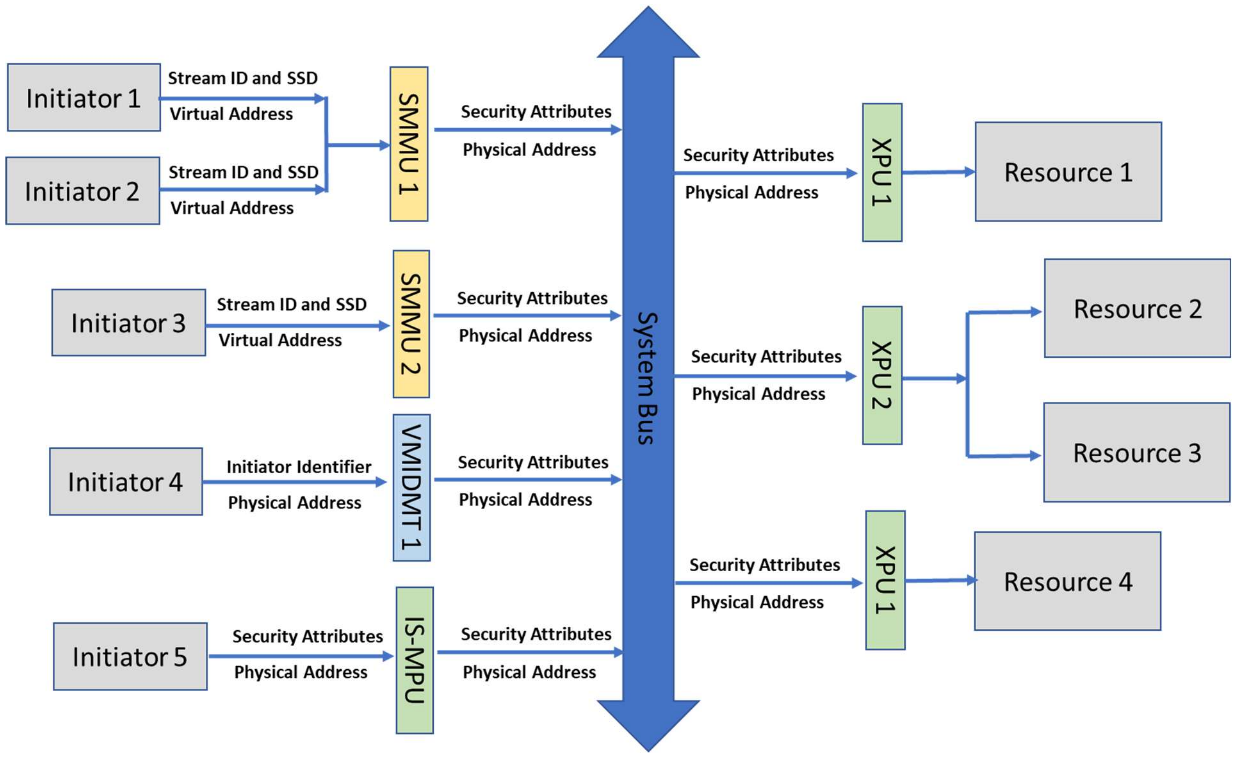

Figure 6: Initiator-side Access Control

An example of initiator-side access control design is shown in Figure 6. There are five initiators and four protected target resources in this example. Transactions from Initiators 1 and 2 go through the same SMMU 1 before reaching the system bus. Initiator 3 is the only initiator connected to SMMU 2. For Initiator 4, there is no SMMU in the path to the System bus and a VMIDMT is used to generate the security attributes to be checked by the XPUs close to the targets. An initiator-side MPU is used to perform permission checking for Initiator 5, and there is no VMIDMT or SMMU in the path from Initiator 5 to the target resources. This design is used in cases where the security attributes of the initiator are known at hardware design time, so they are generated directly by the initiator. On the target-side, there are XPUs in front of Resources 1, 2, 3 and 4.

If Initiator 3 issues a transaction to Resource 1, SMMU 2 performs address translation and access control. If the SMMU allows Initiator 3 to access Resource 1, the security attributes along with other transaction properties will propagate to the system bus, which does address decoding and forwards the transaction towards Resource 1. The XPU 1 in front of Resource 1 performs a separate permission check and allows or rejects the transaction accordingly.

Comparison of Target and Initiator-side Access Control

Initiator-side access control topology is commonly used on newer Snapdragon SoCs. This topology offers more flexibility and scalability to handle new use cases requiring fragmented ownership of, and access to, resources. Target-side topology is more constrained because the XPU instantiation parameters, such as the number of resource groups, are determined at hardware design time. Further, initiator-side access control is more aligned with trends in the ARM ecosystem and industry more broadly.

Access Control Software Configurations

In this section, we will show examples of VMIDMT, XPU, and SMMU software configurations. The access control configurations can be programmed statically during ROT domain software initialization or dynamically when switching from one use case to another.

VMIDMT

An example VMIDMT configuration for a shared Direct Memory Access (DMA) engine is shown in Table 1. There are two channels in this DMA, and the channel identifier serves as the initiator identifier to the VMIDMT. The DMA is shared by two domains and each domain owns one channel. The CPU OS uses channel 0, so the VMID output from channel 0 denotes CPU OS. The CPU OS is in the non-secure domain. TrustZone owns channel 1 so channel 1 outputs the secure signal.

TrustZone configures the VMID and secure signal generation for all channels. It is possible for TrustZone to delegate the configuration of VMID generation to a trusted domain. This delegation can be configured independently for each individual VMIDMT. TrustZone cannot delegate control of how the secure signal is generated to other domains.

Table 1: VMIDMT Configuration Example

Initiator Identifier	VMID Output	Secure Signal
0	CPU OS	Non-secure
1	TrustZone	Secure

XPU

Table 2 shows an example of configurations for an MPU with 2 resource groups. This MPU protects a memory of size 96KB accessed through memory addresses 0x1000_0000 to 0x1001_8000. The start and end address of each MPU resource group must be aligned on a 4K boundary. In this example, TrustZone owns both resource groups. TrustZone programs the start and end addresses, and which initiators can read or write these resource groups. Access to each resource group can be restricted to a single domain or shared among multiple domains.

Table 2: XPU Configuration Example

Resource Group	Start Address (inclusive)	End Address (exclusive)	Size	Owner	Read Permission	Write Permission
0	0x1000_0000	0x1001_0000	64KB	TrustZone	TrustZone CPU OS	TrustZone
1	0x1001_0000	0x1001_8000	32KB	TrustZone	CPU OS	CPU OS

SMMU

Table 1 gives an example SMMU configuration for SMMU1 in Figure 6.

Table 3: SMMU Configuration Example

Stream ID / SSD	Stage 1 Context Bank	Stage 2 Context Bank	Stage 1 ASID	Stage 2 VMID	Secure Signal
0x0000	7	None	TrustZone	None	Secure
0x0001	None	4	None	Audio	Non-secure
0x0100	0	5	Process 1	CPU OS	Non-secure

SMMU1 has 8 context banks (0 to 7) that are used to direct address translation and access control for incoming transactions based on their Stream ID and SSD attributes. For simplicity, the Stream ID signals are also connected as SSD signals in this example.

Initiator1 has two logical channels, 0 and 1, which generate Stream IDs 0x0000 and 0x0001. Initiator 2 generates Stream ID 0x0100. These are assigned to different domains in the SMMU1 configuration below as follows.

Channel 0 is claimed by TrustZone for itself. TrustZone configures SSD 0x0000 to generate secure outgoing transactions, configures context bank 7 as a secure Stage 1 context bank (i.e., further configuration is possible only from TrustZone), and maps Stream ID 0x0000 to context bank 7. TrustZone may attach page tables to context bank 7 to govern address translation and access control. Secure transactions use only single-stage translation, so channel 0 has no associated Stage 2 context bank.

TrustZone configures the remaining SSD values to generate non-secure outgoing transactions, and delegates configuration of the remaining context banks to the Hypervisor.

Channel 1 is assigned to the audio system. The Hypervisor maps Stream ID 0x0001 to context bank 4 and configures this for Stage 2 translation, attaching the Audio domain page tables. In this example, the Audio domain uses Stage-2-only address translation, so the Hypervisor does not assign a Stage 1 context bank to channel 1.

Initiator 2 is assigned to a CPU process. The Hypervisor configures context bank 0 for Stage 1 translation, connects it to context bank 5 for Stage 2 translation, and attaches the CPU OS stage 2 page tables. The Hypervisor delegates configuration of Stage 1 translation to the CPU OS. The Hypervisor leaves the remaining context banks 1, 2, 3 and 6 as inactive. Note that the Hypervisor delegates only specific Stream IDs and context banks for configuration by the CPU OS.

Access Control Use Case Example

In this section, we will step through an example access control use case – firmware authentication. The purpose of firmware authentication is to prevent unauthorized code from being executed on the device. The scenario is that TrustZone does not have direct access to the external flash drive and must rely on the CPU OS to load the Video CPU firmware from external storage to DRAM. However, the CPU OS is not trusted by Video CPU and, therefore, should not have access to Video firmware memory while the Video CPU is running. Figure 7 shows the relevant software flow.

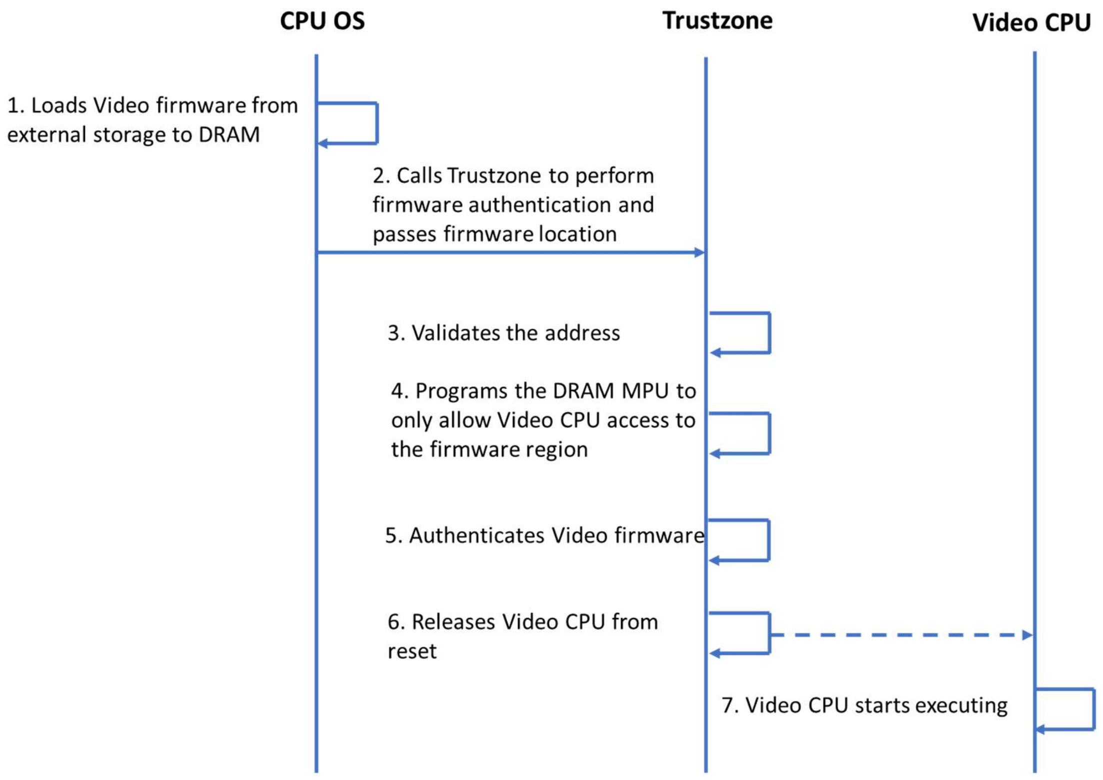
CPU OS
Trustzone
Video CPU
1. Loads Video firmware from external storage to DRAM
2. Calls Trustzone to perform firmware authentication and passes firmware location
3. Validates the address
4. Programs the DRAM MPU to only allow Video CPU access to the firmware region
5. Authenticates Video firmware
6. Releases Video CPU from reset
7. Video CPU starts executing

Figure 7: Firmware Authentication Software Flow

1. The CPU OS loads the video firmware from external storage to DRAM. At this point, the CPU OS has full access to the DRAM memory holding the Video firmware and can modify it at will.
2. The CPU OS calls TrustZone to perform firmware authentication, passing the address and size of the loaded Video firmware.

3. TrustZone validates the address range to ensure there is no overlapping with other use cases. An error is returned to the CPU OS if verification fails.
4. TrustZone programs one DRAM MPU resource group to protect Video firmware address range, granting read and write permission for this resource group to Video CPU only. After this step, the CPU OS is no longer able to read or modify the Video firmware.
5. TrustZone authenticates the firmware.
6. If the authentication passes, TrustZone releases Video CPU from reset.
7. Video CPU starts executing.

If the authentication fails, the MPU resource group for Video CPU is released and an error is returned to the CPU OS.

The access control protection provided in step 4 is essential to the firmware authentication use case. If the Video firmware region remained open, any domain could modify the Video code in DRAM, defeating the goal of firmware authentication. It is also worth noting that the Video CPU reset control registers must also be protected. If not, any domain could release Video CPU from reset and let it run arbitrary code.

Summary

Access control is one of the most important building blocks of SoC security architecture. We have provided an overview of how access control works on Snapdragon SoCs. The three access control hardware components –VMIDMT, XPU, and SMMU – are programmed by software, and the resulting policies are enforced by the hardware. These components can be placed in a target-side or initiator-side topology. Older SoCs tended toward target-side access control topology while more modern SoCs use initiator-side topology. Finally, we illustrated the critical role of access control in secure use cases through the example of firmware authentication.

References

[1] "ARM TrustZone," [Online]. Available: https://developer.arm.com/ip-products/security-ip/TrustZone.

[2] "ARM Virtualization," [Online]. Available: https://developer.arm.com/docs/100942/latest/aarch64-virtualization.

[3] "ARM System Memory Management Units," [Online]. Available: https://developer.arm.com/ip-products/system-ip/system-controllers/system-memory-management-unit.