[Trustzone]-Arm Trustzone 的安全擴展介紹-一篇就夠了

PSATrustZone

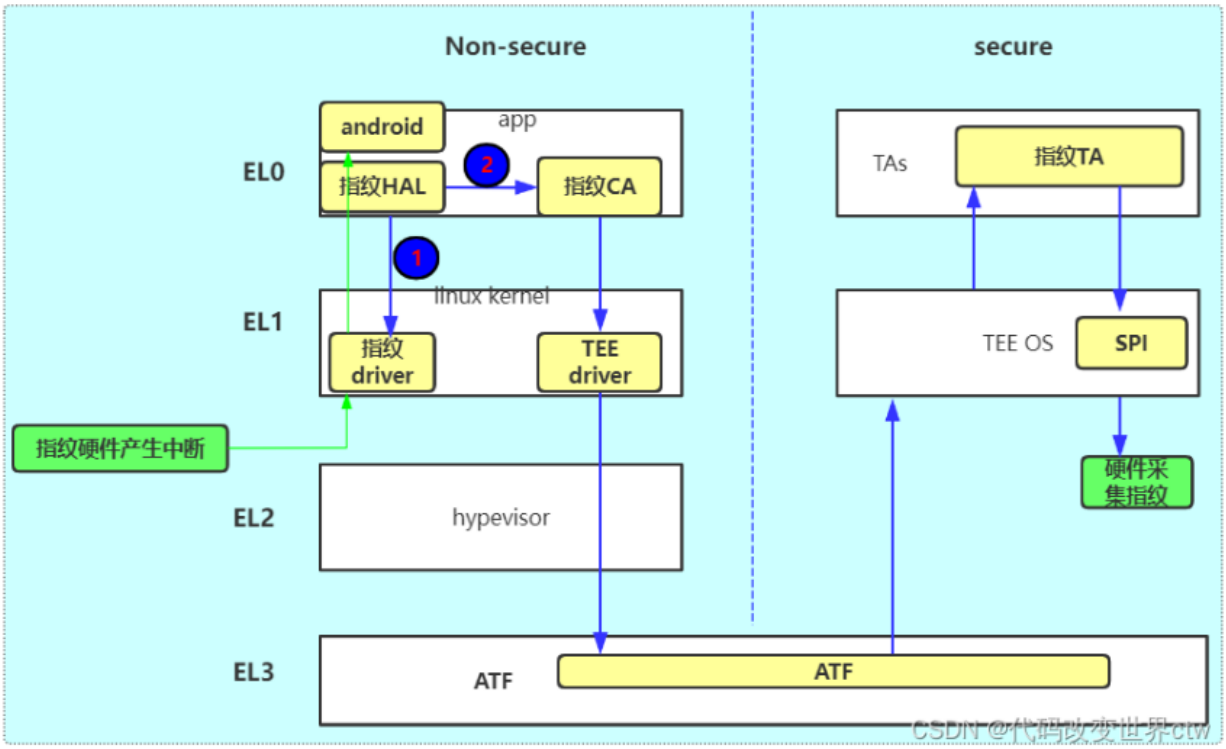
1、背景：

隨著時代的發展、科技的進步，安全需求的趨勢也越來越明顯，ARM 也一直在調整和更新其新架構，很多都是和安全相關的。

如下列出了一些和安全相關的架構

Trustzone 做為 ARM 安全架構的一部分，從 2008 年 12 月 ARM 公司第一次 release Trustzone 技術白皮書。(trustzone white paper – ARM Trustzone 安全白皮書百度網盤下載，密碼:1234)

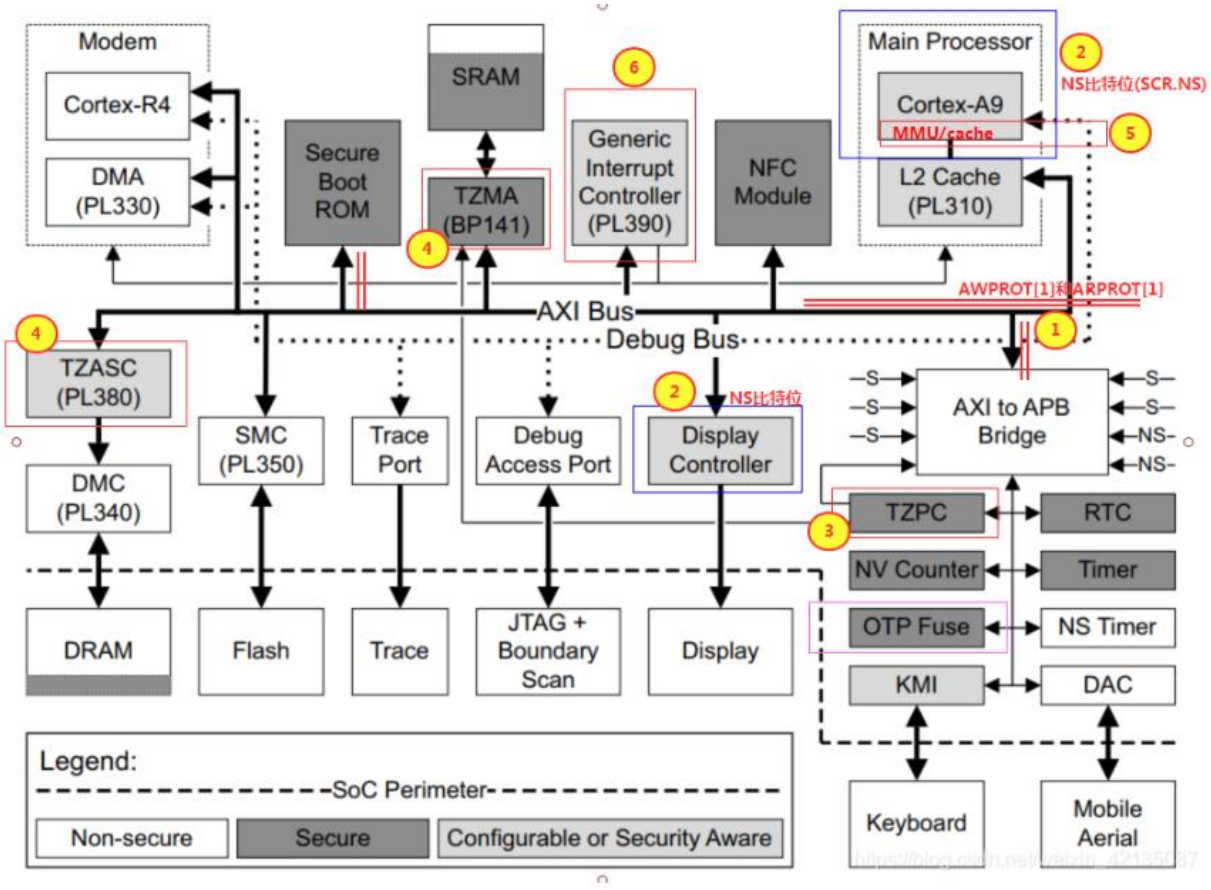
2013 年 Apple 推出了第一款搭載指紋解鎖的 iPhone：iPhone 5s，用以保證指紋資訊安全的 Secure Enclave 技術據分析深度定製了 ARM trustzone 架構，印象中這大概是 Trustzone 技術第一次走進大眾視線。到如今 Trustzone 技術已經成為移動安全領域的重要基礎技術，你也許不瞭解它的技術原理，但它一直默默為你守護你的指紋資訊，帳戶密碼等各種敏感資料。 如下也列出了一張在 Trustzone 架構下的一張指紋的框圖，這也是這些年(2015-至今)比較流行的一張軟體框圖。



2、ARM Trustzone 的安全擴展簡介

從上文我們已經知道， ARM Trustzone 不具體指一個硬體，也不是一個軟體，而是一個技術架構，在支援 ARM Trustzone 的 SOC 中，需按照 ARM Trustzone 技術對各個

子模組進行設計。如下便展示了一個 SOC 的 Trustzone 架構下的設計框圖



其中：

(1)、AMBA-AXI 匯流排的擴展，增加了標誌 secure 讀和寫地址線：AWPROT[1]和 ARPROT[1]

(2)、processor 的擴展(或者說 master 的擴展)，在 ARM Core 內部增加了 SCR.NS 位元位，這樣 ARM Core 發起的操作就可以被標記“是以 secure 身份發起的訪問，還是以 non-secure 身份發起的訪問”

(3)、TZPC 擴展，在 AXI-TO-APB 端增加了 TZPC，用於組態 apb controller 的權限(或者叫 secure controller)，例如將 efuse(OTP Fuse)組態成安全屬性後，那麼 processor 以 non-secure 發起的訪問將會被拒絕，非法的訪問將會返回給 AXI 匯流排一個錯誤。

(4)、TZASC 擴展，在 DDRC(DMC)之上增加一個 memory filter，現在一般都是使用 TZC400，或由 SOC 廠商自己設計一個這樣的 IP，或叫 MPU，或整合在 DMC 內部，它的作用一般就是組態 DDR 的權限。 如果組態了 DDR 中某塊 region 為安全屬性，那麼 processor 以 non-secure 發起的訪問將會被拒絕。

(5)、MMU/Cache 對安全擴展的支援

在軟體架構的設計中，就分為: Non-secure EL0&1 Transslation Regime 和 Secure EL0&1 Transslation Regime，即 normal world 和 secure world 側使用不同的 Transslation Regime，其實就是使用不同的 TTBRx_ELn 暫存器，使用不同得頁表。

注意：在 armv7 上，TTBRx_EL0、TTBRx_EL1 是 banked by Security State，也就是說在安全世界和非安全世界各有一組這樣的暫存器，所以在 linux 和 tee 中可以各自維護一張自己的記憶體頁表.

在 armv8/armv9 上，TTBRx_EL0、TTBRx_EL1 不再是 banked 了，但是 world switch 時會在 ATF 中 switch cpu context, 所以從 hypervisror 或 os 的視角來看，依然還是兩套不同的 TTBRx_ELn 暫存器，linux 和 tee 各有各的頁表。

而在 TLB 中，又為每一個 entry 增加了 Non-secure 屬性位，即標記當前翻譯出的實體位址是 secure 還是 non-secure；

cache 的擴展：在 cache 的 entry 中的 TAG 中，有一個 NON-Secure Identifier 標記為，表示當前快取資料的實體位址是屬於 non-secure 還是 secure。

(6)、gic 對安全擴展的支援，在 gicv2、gicv3 的版本中，都增加了對安全擴展的支援. 以 gicv3 為例，將中斷劃分成了 group0、secure group1 和 non-secure group1. 在軟體的組態下，group0 和 secure group1 的中斷將不會 target 到 REE(linux)中處理

3、ARM Trustzone 的安全擴展詳細解剖

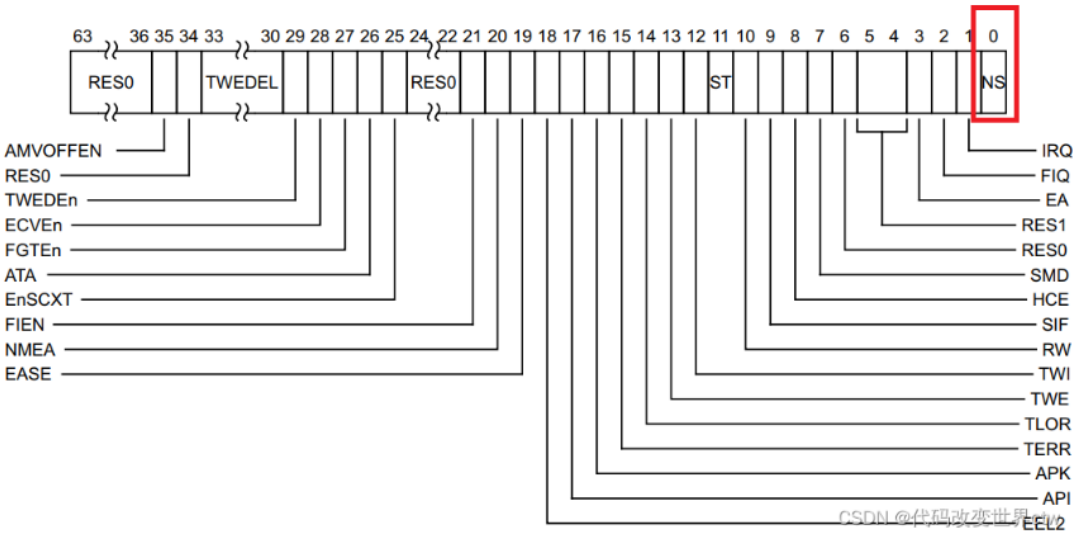
3.1 AMBA-AXI 對 Trustzone 的支援

ARPROT[2:0]和 AWPROT[2:0] 分別是讀通道和寫通道中的關於權限的訊號，例如他們中的 BIT[1]則分別表示正是進行 secure 身份的讀或 secure 身份的寫操作。

AxPROT	Value	Function
[0]	0	Unprivileged access
	1	Privileged access
[1]	0	Secure access
	1	Non-secure access
[2]	0	Data access
	1	Instruction access

3.2 Processor 的 SCR.NS 位元位

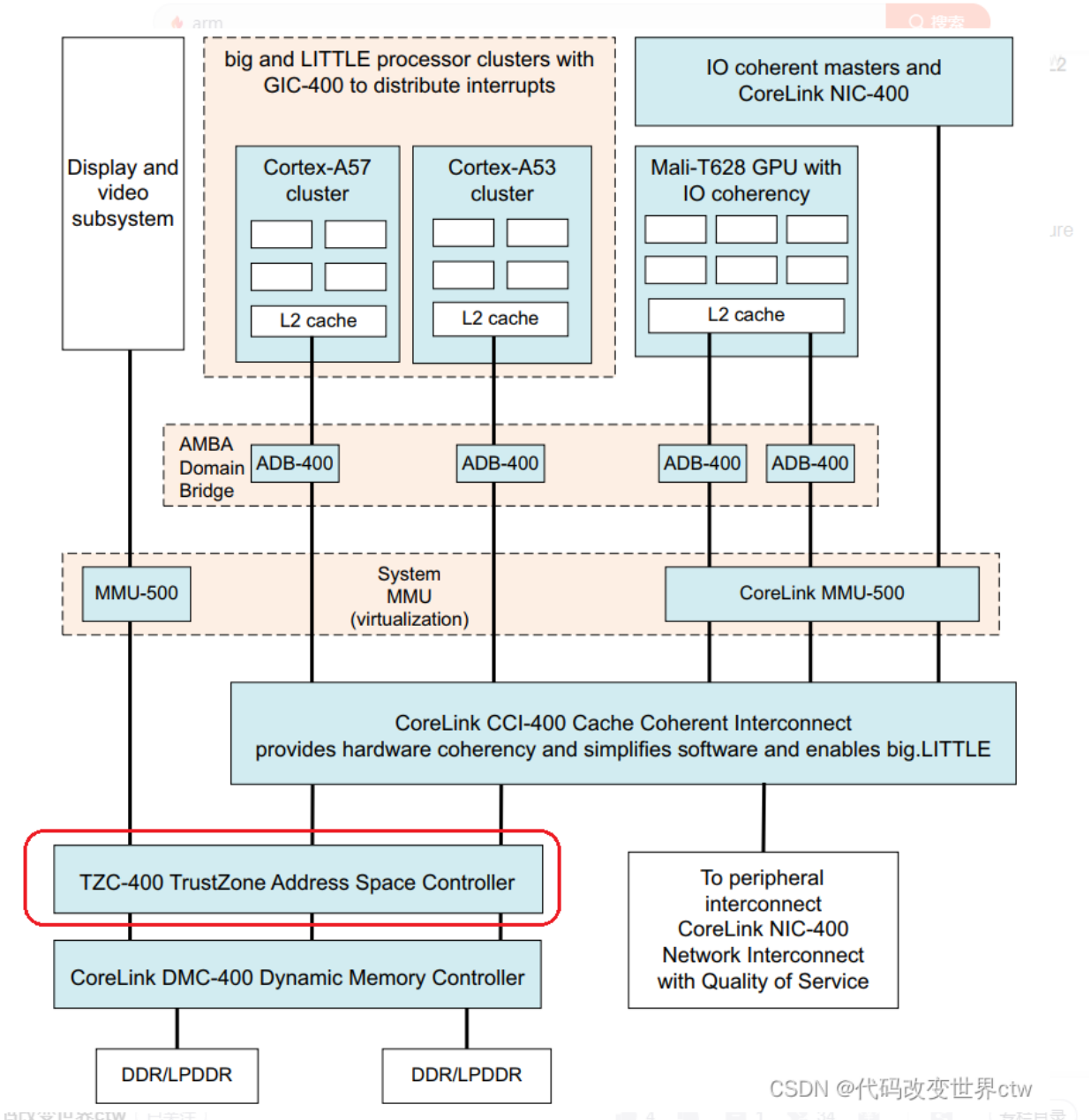
SCR_EL3.NS 表示當前 processor 的安全狀態，NS=1 表示是 non-secure 的，NS=0 表示是 Secure 的



3.3 TZC400 和 TZPC 簡介

TZC400 接在 core 和(DMC)DDR 之間，相當於一個 memory filter。

TZC400 一般可以組態 8 個 region（算上特殊 region0, 也可以說 9 個），然後可以對每一個 region 組態權限。例如講一塊 region 組態成 secure RW 的，那麼當有 non-secure 的 master 來訪問這塊記憶體時，將會被 TZC 擋住。

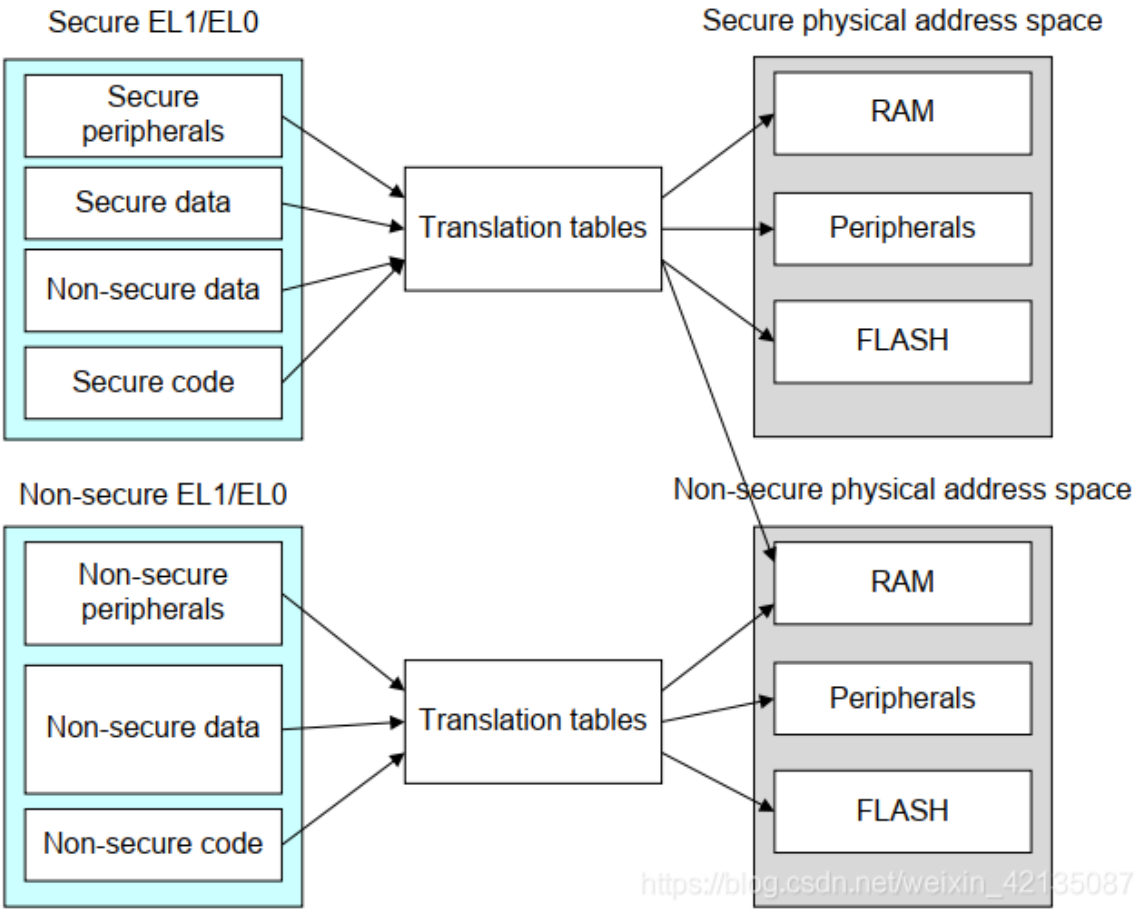


3.4 MMU 對 Trustzone 的支援

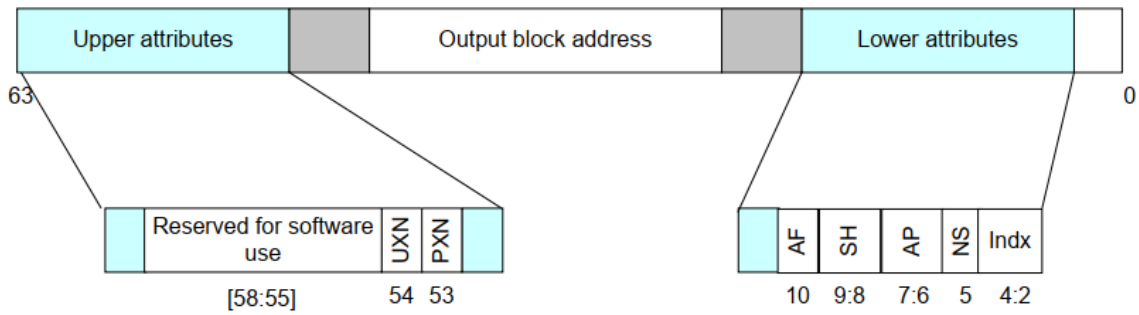
首頁，在軟體架構的設計中，就分為: Non-secure EL0&1 Transslation Regime 和 Secure EL0&1 Transslation Regime，即 normal world 和 secure world 側使用不同的 Transslation Regime，其實就是使用不同的 TTBRx_ELn 暫存器，使用不同得頁表

其次，在 MMU 使用的頁表中，也有 NS 位元位。 Non-secure Transslation Regime 只能翻譯 NS=1 的頁表項，secure Transslation Regime 可以翻譯 NS=1 和 NS=0 的

頁表項。即 secure 的頁表可以對應 non-secure 或 secure 的記憶體，而 non-secure 的頁表只能去對應 non-secure 的記憶體，否則在轉換時會發生錯誤



在 Page Descriptor 中(頁表 entry 中)，有 NS 位元位（BIT[5]），表示當前的對應的記憶體屬於安全記憶體還是非安全記憶體：



- Unprivileged eXecute Never (UXN) and Privileged eXecute Never (PXN) are execution permissions.
- AF is the access flag.
- SH is the shareable attribute.

3.5 cache 對 Trustzone 的支援

如下所示，以為 cortex-A78 為例，L1 Data Cache TAG 中 ，有一個 NS 位元位（BIT[33]），表示當前快取的 cacheline 是 secure 的還是 non-secure 的

Bit field	Description
[63:41]	0
[40:34]	ECC
[33]	Non-secure identifier for the physical address
[32:5]	Physical address [39:12]
[4:3]	Reserved
[2]	Transient/WBNA
[1:0]	MESI **00** Invalid **01** Shared **10** Exclusive **11** Modified with respect to the L2 cache

3.6 TLB 對 Trustzone 的支援

如下所示，以為 cortex-A78 為例，L1 Data TLB entry 中 ，有一個 NS 位元位（BIT[35]），表示當前快取的 entry 是 secure 的還是 non-secure 的

L1 data TLB cache format for data register 0

Bit field	Description
[63:62]	Virtual address [13:12]
[58]	Outer-shared
[57]	Inner-shared
[52:50]	Memory attributes: **000** Device nGnRnE **001** Device nGnRE **010** Device nGRE **011** Device GRE **100** Non-cacheable **101** Write-Back No-Allocate **110** Write-Back Transient **111** Write-Back Read-Allocate and Write-Allocate
[38:36]	Page size: **000** 4KB **001** 16KB **010** 64KB **011** 256KB **100** 2MB **101** Reserved **110** 512MB **111** Reserved
[35]	Non-secure
[34:33]	Translation regime: **00** Secure EL1/EL0 **01** Secure EL3 **10** Non-secure EL1/EL0 **11** Non-secure EL2
[32:17]	ASID
[16:1]	VMID
[0]	Valid

3.7 gicv 的安全中斷

在 gicv2/gicv3 中，支援了安全中斷，組態有如下：

（1）、Group 分組(GICD_IGROUPRn) – gicv2

- group0：安全中斷，由 nFIQ 驅動
- group1：非安全中斷，由 nIRQ 驅動

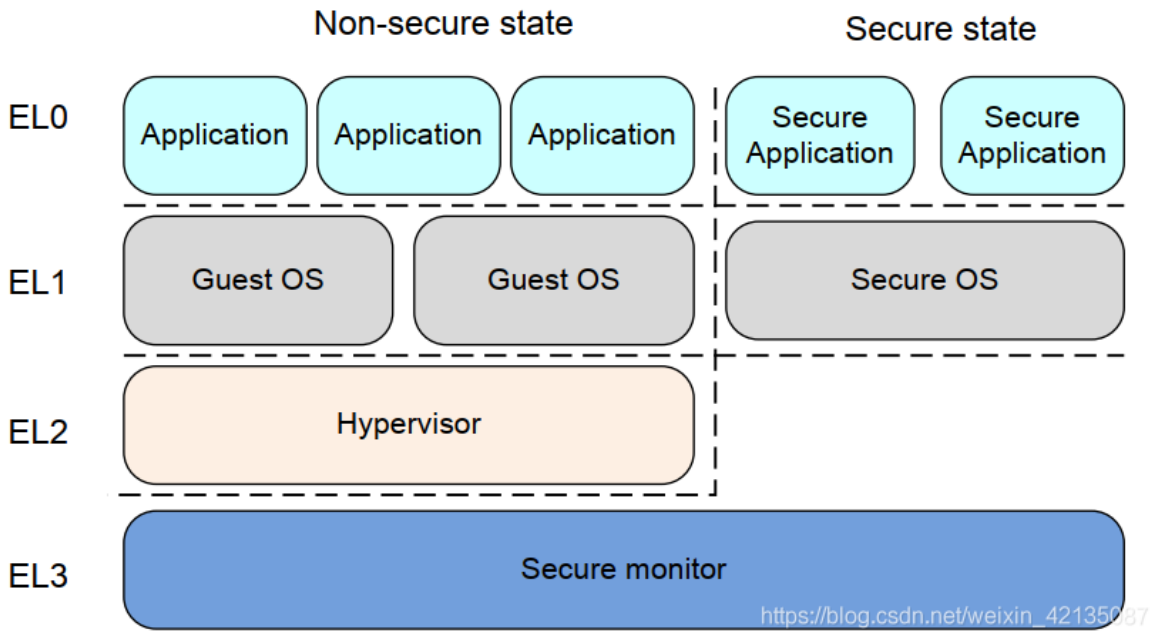
（2）、Group 分組(GICD_IGROUPRn)– gicv3

- group0：安全中斷
- non-secure group1：非安全中斷
- secure group1：安全中斷

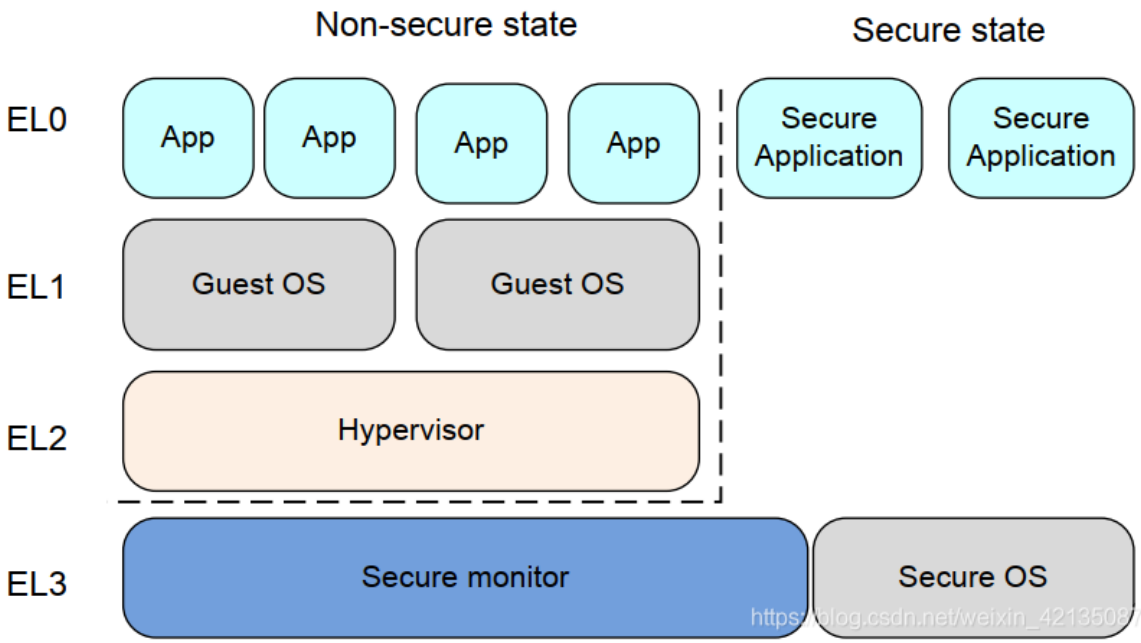
4、ARM Trustzone 技術對軟體帶來的變化

ARM Trustzone 技術對軟體框架帶來了變化

4.1、EL3 is AArch64：



4.2、EL3 is AArch32：

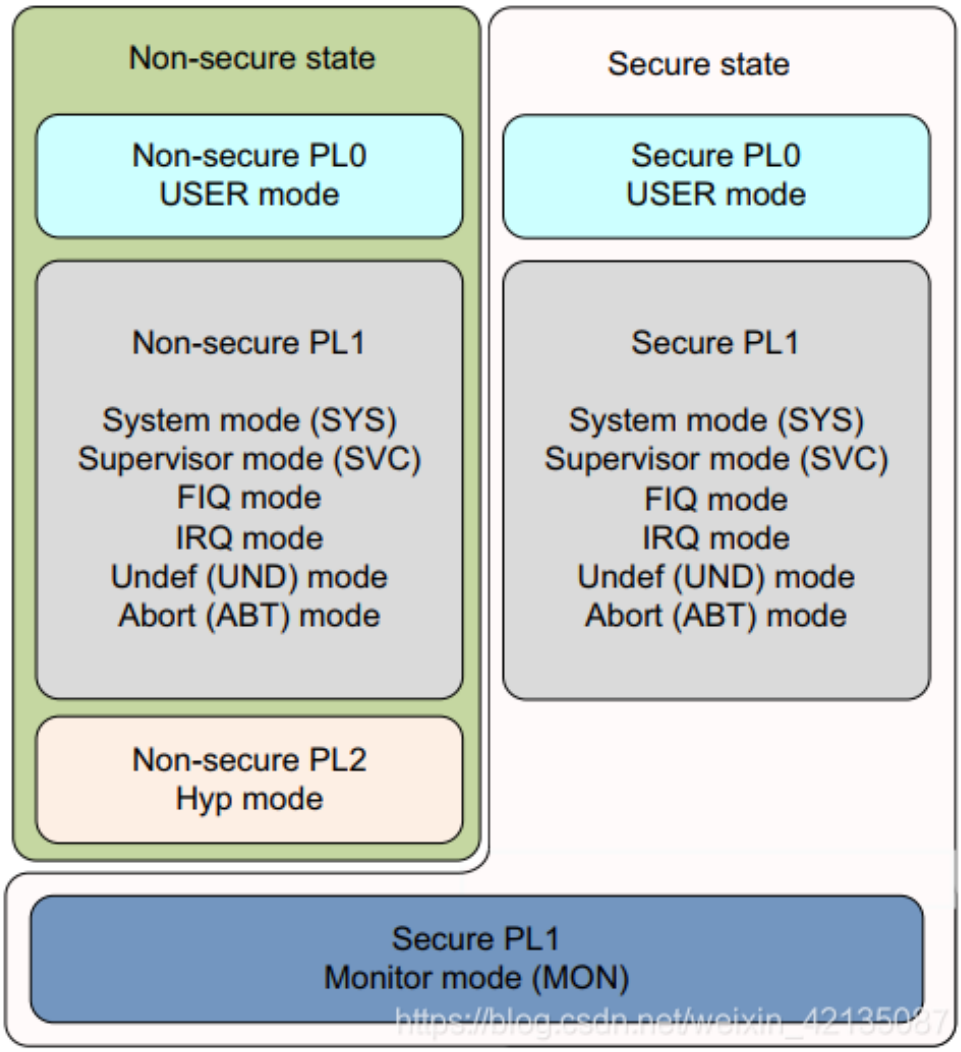


AArch32 和 AArch64 secure monitor 的理解：

如果 secureos 和 monitor 都是 64 位，secureos 跑在 el1, monitor 跑在 el3;
如果 secureos 和 monitor 都是 32 位，secureos 和 monitor 都跑在 EL3（secureos 在 svc 模式、monitor 在 svc 模式），它倆共用頁表；

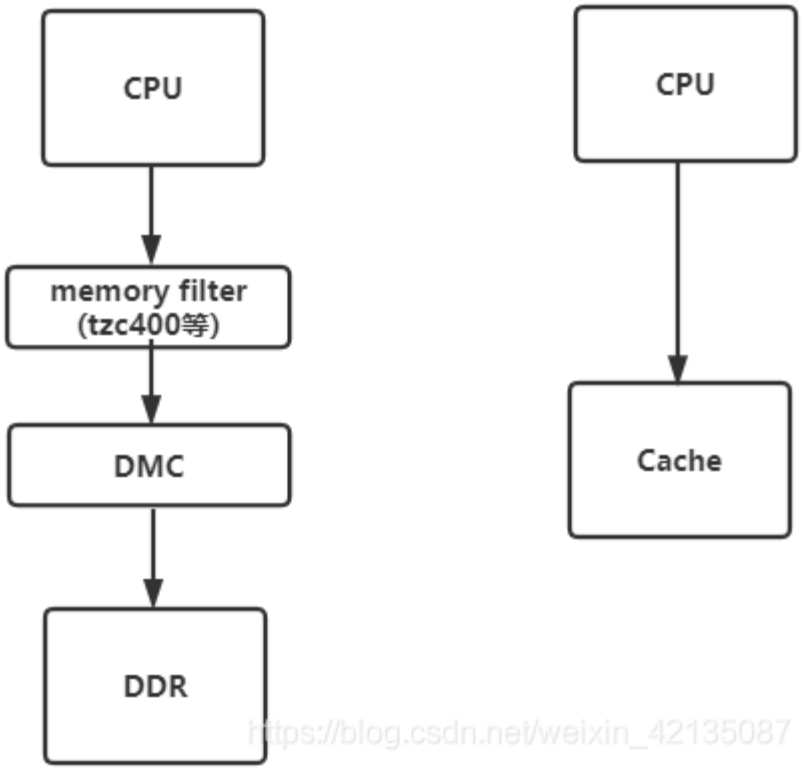
如果 monitor 是 64 位，secureos 是 32 位，那麼 secureos 跑在 svc 模式(el1)，monitor 跑在 el3，他倆不共用頁表

4.3、armv7：

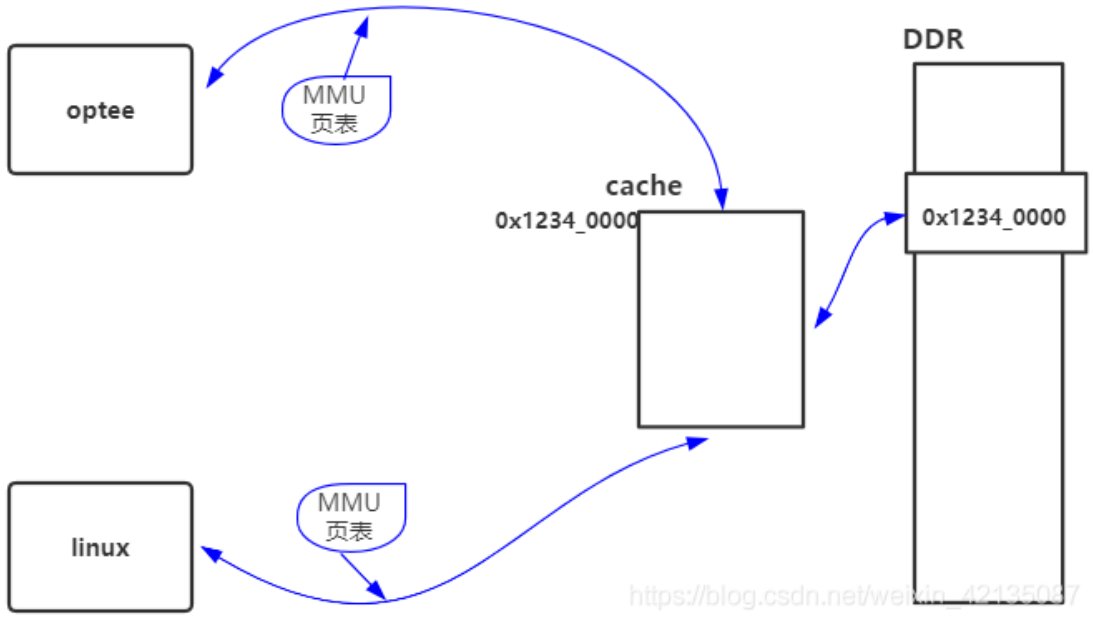


5、思考：通過 MMU/TLB/Cache 對安全記憶體攻擊的可能性

在安全架構的設計時，我們在 Core 和 DDR 之間增加了一個 TZC 做為 memory filter，資料流為:Core ---> TZC---->DDR, 這種架構下，core 以非安全身份發起的對安全記憶體的讀寫，將會被 TZC 擋住。



但是這都是在理想的情況下，事實上 Core 發起對記憶體的讀寫，未必經過 TZC 未必到 DDR，有可能到 cache 階段就完成了，即資料流變成了 Core ---> MMU(TLB+Addtress Translation) ---->Cache，那麼這種情況下，沒有 TZC 的事了，你也許會說 MMU/Cache 中都有 NS 位元，但是你真的理解這裡 NS 位元的用法嗎？如果 core 以非安全身份對安全記憶體發起的讀寫時，我強制將 MMU 頁表中的安全屬性標記位強制改成 NS=0，會如何呢？



事實上我們只要理清原理、理清資料流 ，就不會問上面那麼 S13 的問題了。 下面來開始剖析:

假設一個安全 core 讀取了一個安全實體記憶體 0x2000_0000 資料(虛擬地址可能是 0x_xxxx_xxxx)，那麼將產生一下行為：

在讀寫之前，勢必做好了 MMU map，如實體位址 0x2000_0000 MAP 成了 0x_xxxx_xxxx 地址, 此時 Page Descriptor 中的 atrribute 中的 NS=0

TLB 快取該翻譯，即 TLB 的 entry 中包含: 0x2000_0000、0x_xxxx_xxxx、NS=0 安全記憶體 0x2000_0000 資料將會被快取到 cache 中，entry 中的 TAG 包含 0x2000_0000、NS=0

同時，我有一個非安全 core 發起讀寫虛擬地址 0x_yyyy_yyyy，我自行修改該頁表，讓 0x_yyyy_yyyy 強制對應到安全實體記憶體 0x2000_0000，此時有兩種組態:

(1)、0x_yyyy_yyyy—0x2000_0000, NS=0

(2)、0x_yyyy_yyyy—0x2000_0000, NS=1

我們分別看下這兩種組態，是否能讀到安全記憶體：

針對(1)，非安全的 core 發起訪問，發現 TLB 中的條目是 0x_yyyy_yyyy—0x2000_0000, NS=0,自然不會被命中，然後使用 Address Translation 轉換，MMU 發現非安全的 Core 要來訪問安全屬性 NS=0 將會被直接拒絕掉。

針對(2)，非安全的 core 發起訪問，由於 NS=1，TLB 可能會被命中，即能翻譯出 0x2000_0000 實體位址來，即使沒有被命中，在經過 Address Translation 轉換，由於 NS=1，此時也是可以正確轉換出正確的 0x2000_0000 實體位址。 然後接著會去 cache 中查詢這個地址，但是此時 cache 的 entry 中的 NS=0，所以 cache 不會被命中，接下來就要走 TZC 流程了，很顯然，你一個非安全的 core 想訪問安全的記憶體，TZC 將會擋住你。

綜上所述：安全就是安全，不要再瞎想漏洞了。

作者：程式碼改變世界 ctw
原文連結：
https://blog.csdn.net/weixin_42135087/article/details/109272384