[[Trustzone]-Arm Trustzone的安全擴展介紹-一篇就夠了](https://aijishu.com/a/1060000000320633)

[PSA](https://aijishu.com/t/psa)[TrustZone](https://aijishu.com/t/trustzone)

**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

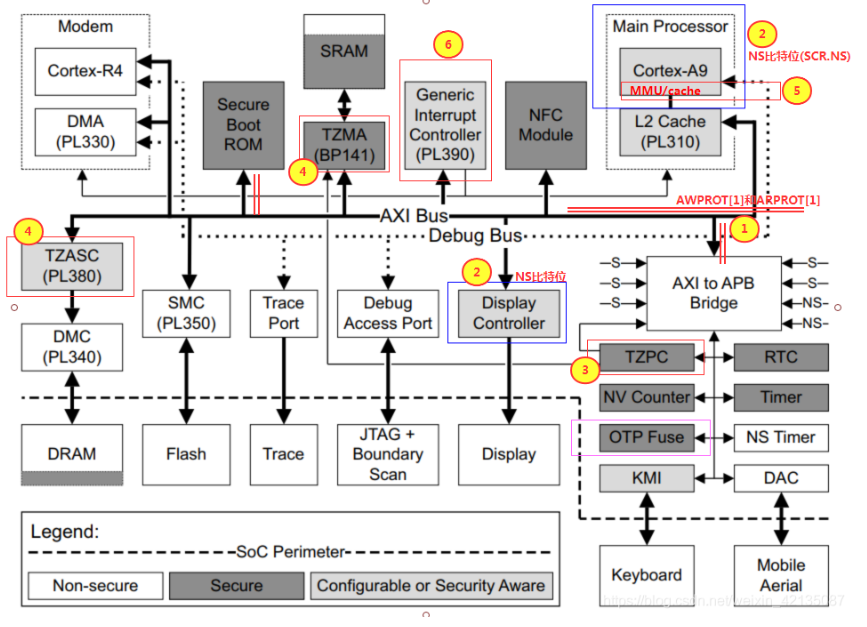
Trustzone做為ARM安全架構的一部分，從 2008 年 12月 ARM 公司第一次 release Trustzone 技術白皮書。(trustzone white paper – ARM Trustzone安全白皮書百度網盤下載, 密碼:1234)

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。

一張含有 圖表 的圖片

自動產生的描述

**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。一張含有 資料表 的圖片

自動產生的描述

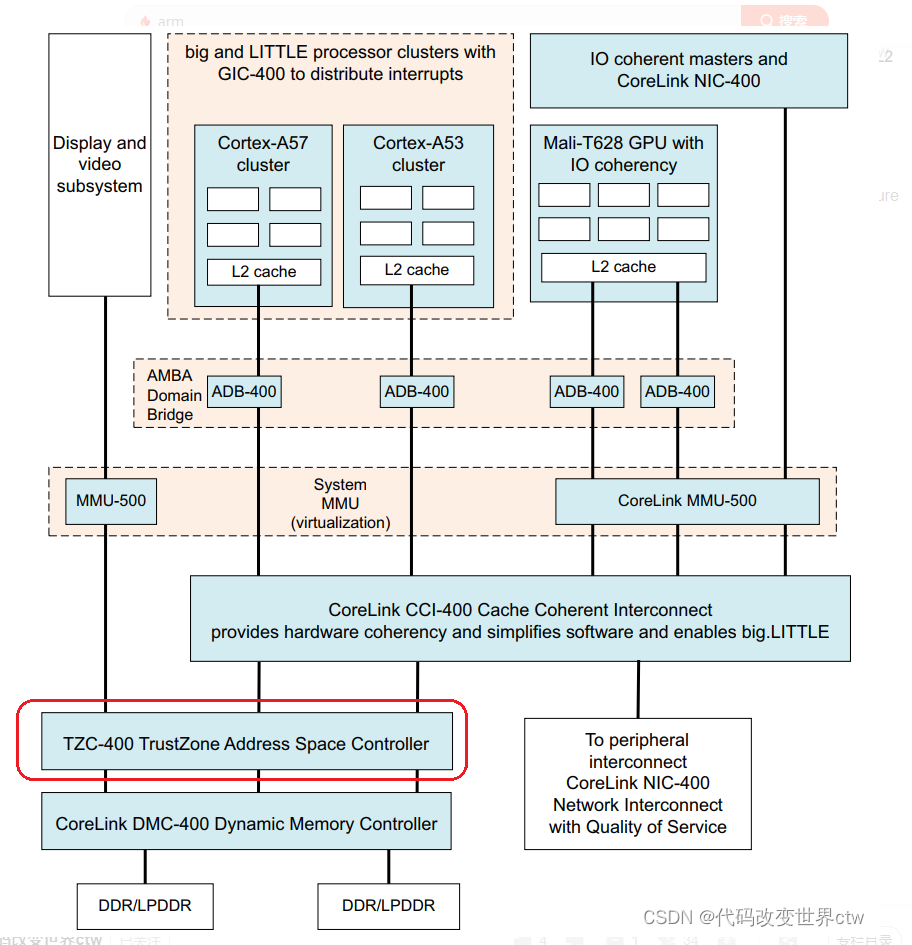
**3.2 Processor的SCR.NS位元位**

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的一張含有 圖表 的圖片

自動產生的描述

**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。

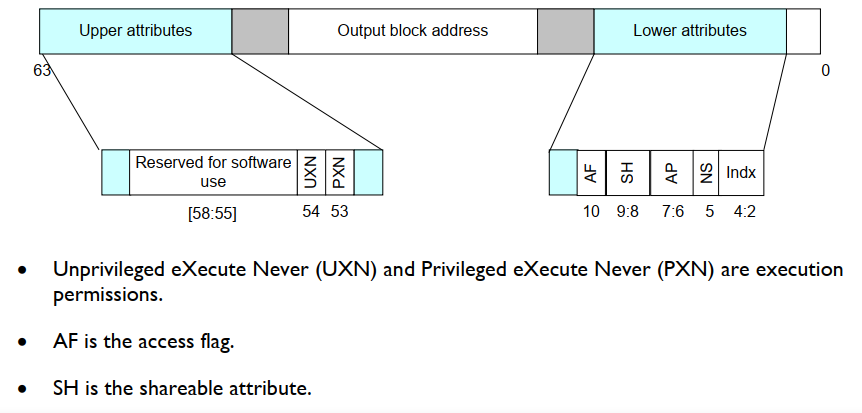
**3.4 MMU對Trustzone的支援**

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

一張含有 圖表 的圖片

自動產生的描述

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：

**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的一張含有 文字 的圖片

自動產生的描述

**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

一張含有 資料表 的圖片

自動產生的描述

**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

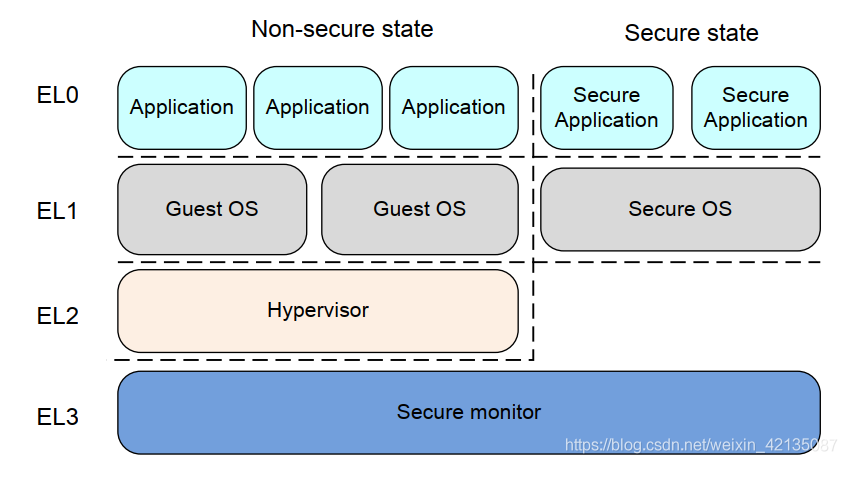
（2）、Group分組(GICD\_IGROUPRn)– gicv3

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

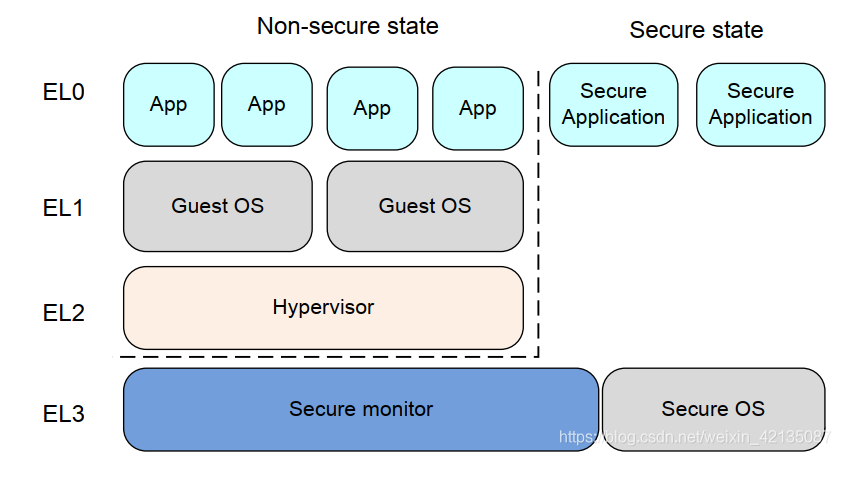
**4、ARM Trustzone技術對軟體帶來的變化**

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：**

一張含有 圖表, 文字 的圖片

自動產生的描述

**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。

一張含有 圖表 的圖片

自動產生的描述

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？

一張含有 圖表 的圖片

自動產生的描述

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](https://aijishu.com/link?target=https%3A%2F%2Fblog.csdn.net%2Fweixin_42135087%2Farticle%2Fdetails%2F109272384)