Аппаратная поддержка доверенной среды исполнения в микроконтроллерах и микропроцессорах с архитектурой ARMv.8A и ARMv.8M

Андрей Самоделов
Системный аналитик
«Лаборатория Касперского»

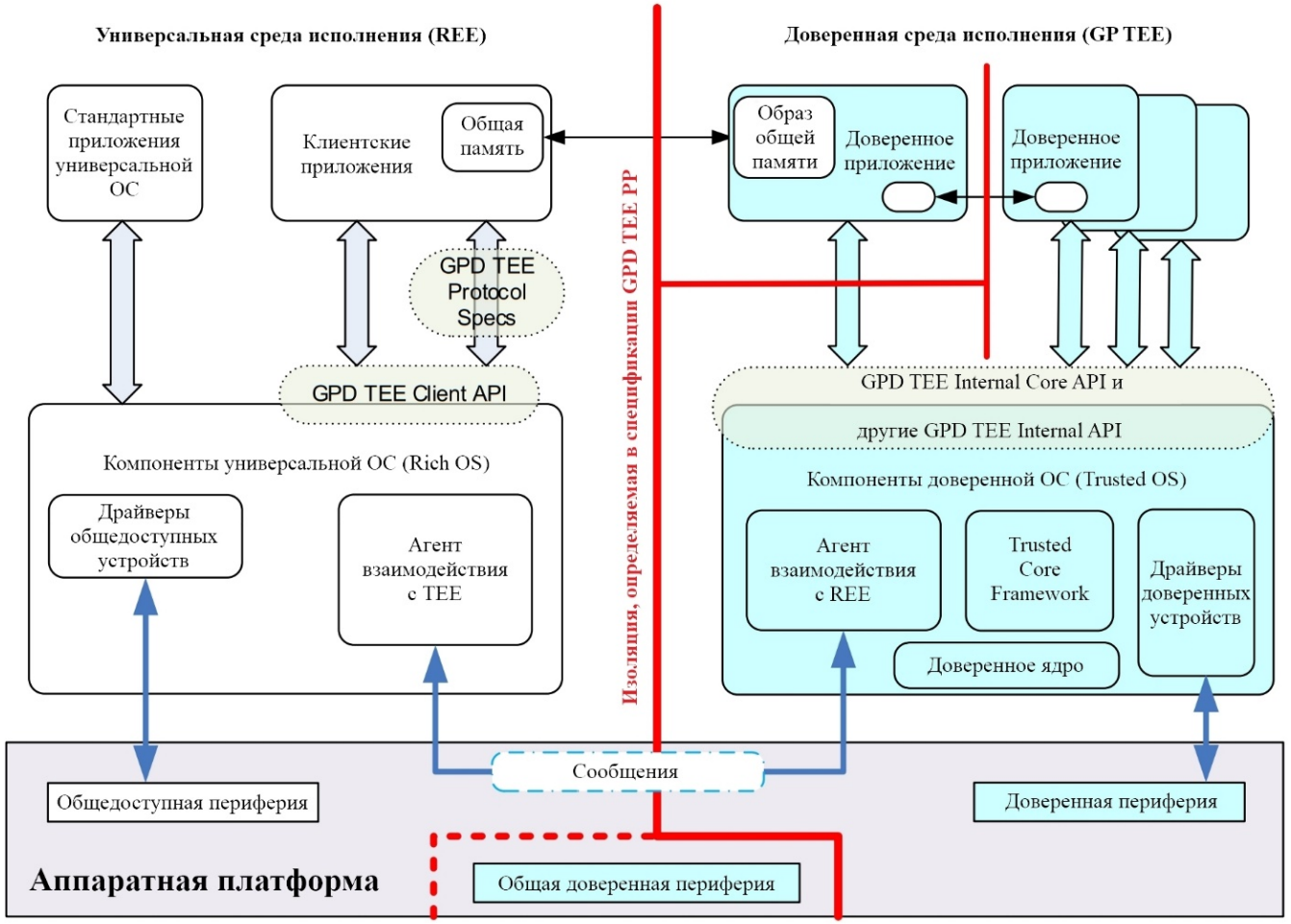
kaspersky

Содержание

kaspersky

Высокоуровневая архитектура доверенной среды исполнения (TEE)

Программная архитектура TEE



Основной задачей доверенной среды исполнения (TEE) является изоляция приложений и аппаратных модулей системы общего назначения от таковых для выполнения критичных операций, например, обработки персональных данных или выполнения криптографических операций, например, формирование электронной подписи под финансовыми документами.

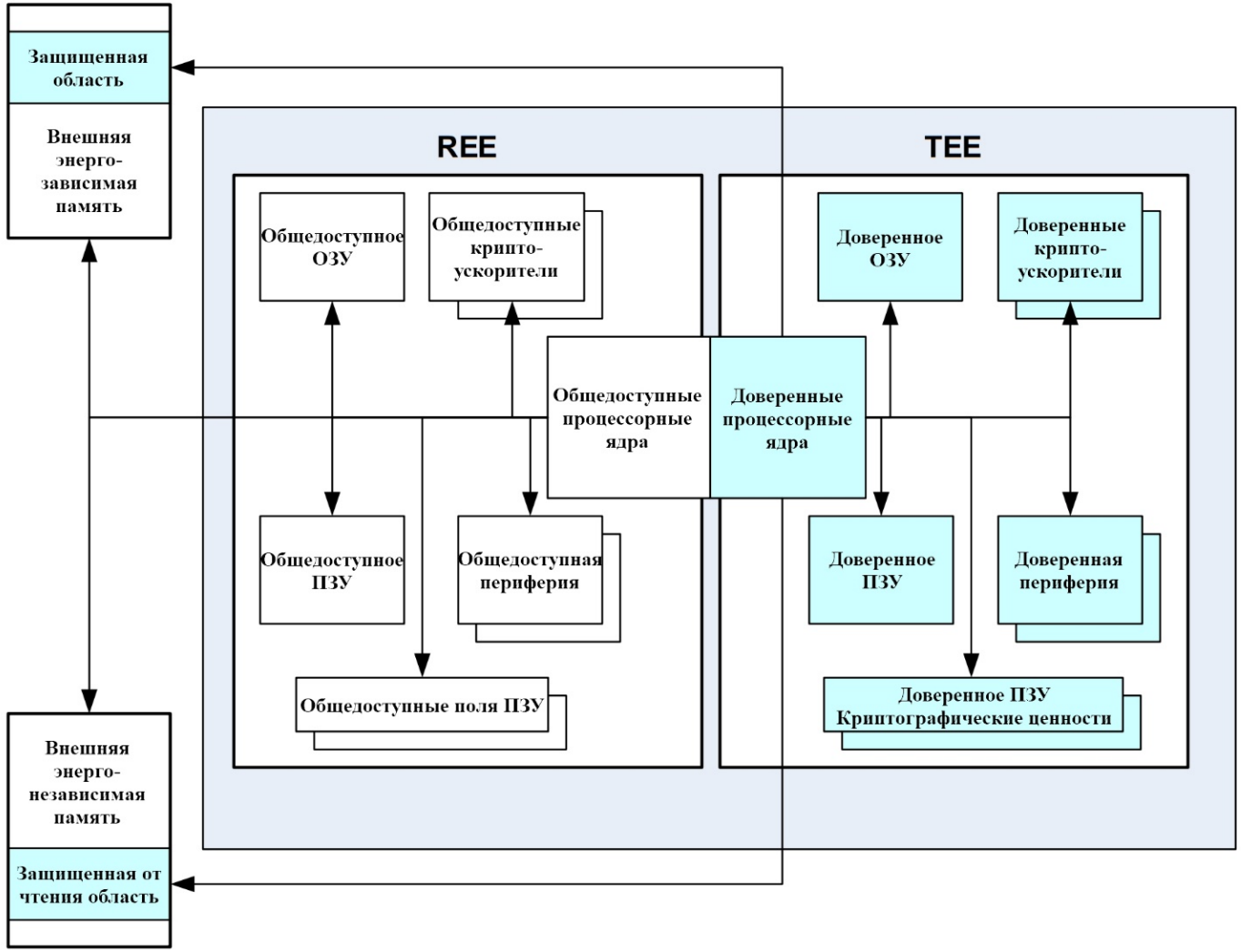
Разделение системы на доверенную (TEE) и универсальную среду исполнения (REE) происходит на нескольких уровнях.

На программном уровне осуществляется:

- Разделение исполняемого кода между средами исполнения на доверенные и недоверенные приложения
- Изоляция контекста исполнения доверенных приложений
- Поддержка на уровне драйверов работы с доверенными периферийными модулями

kaspersky

Аппаратная архитектура TEE

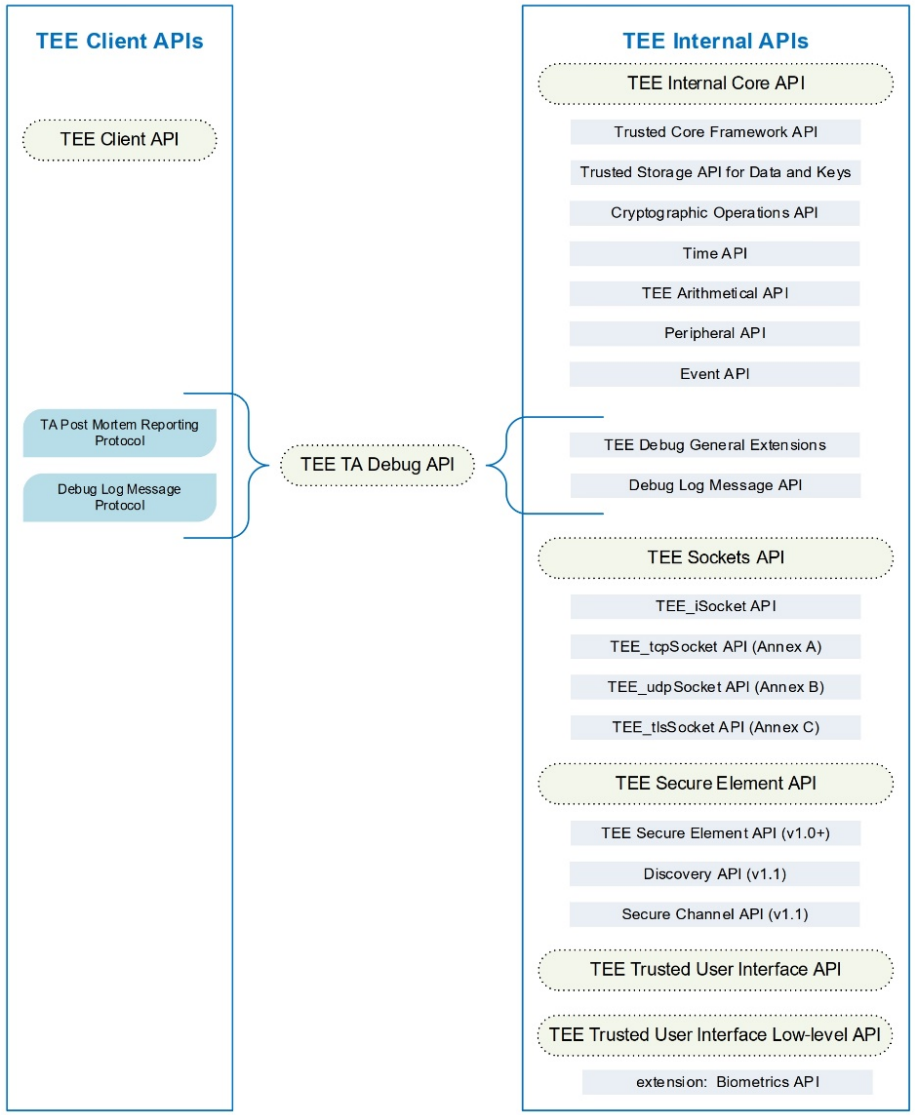


Для максимальной изоляции доверенной среды исполнения (TEE) от общей среды исполнения (REE) кроме программной необходима аппаратная поддержка, которая должна обеспечивать:

- Разделение процессорных ядер на исполняющие доверенный код и исполняющие код среды общего назначения
- Разделение общего ОЗУ системы на доверенную и общедоступную области
- Разделение общего внутреннего ПЗУ системы на доверенную и общедоступную области
- Разделение периферийных модулей (в том числе крипто-ускорителей) на доверенные и общедоступные
- Наличие доверенного хранилища ценностей, в том числе криптографических

kaspersky

Интерфейсы прикладного программирования - TEE API



Для взаимодействия между общей средой исполнения (REE) и доверенной средой исполнения (TEE) разработан ряд интерфейсов прикладного программирования - TEE API, которые делятся на 2 основных части:

- TEE Client API
- TEE Internal API

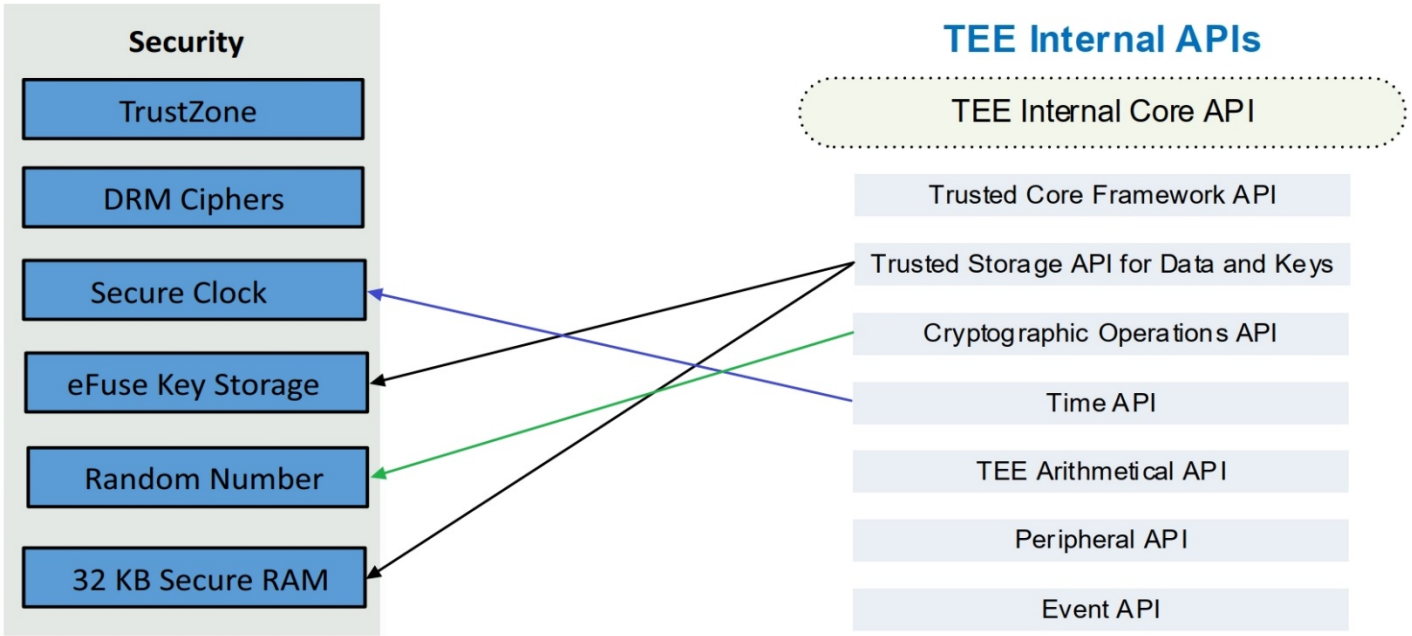
TEE Client API служит для организации взаимодействия между недоверенными приложениями REE и доверенными приложениями TEE

TEE Internal API делится на несколько больших категорий:

- TEE Internal Core API, предназначенный для обеспечения жизненного цикла доверенных приложений, в том числе криптографических
- TEE Socket API, предназначенный для защищенного обмена данными
- TEE Secure Element API, предназначенный для организации корня доверия системы
- TEE Trusted User Interface API, предназначенный для реализации защищенного пользовательского интерфейса

kaspersky

Аппаратная поддержка TEE API

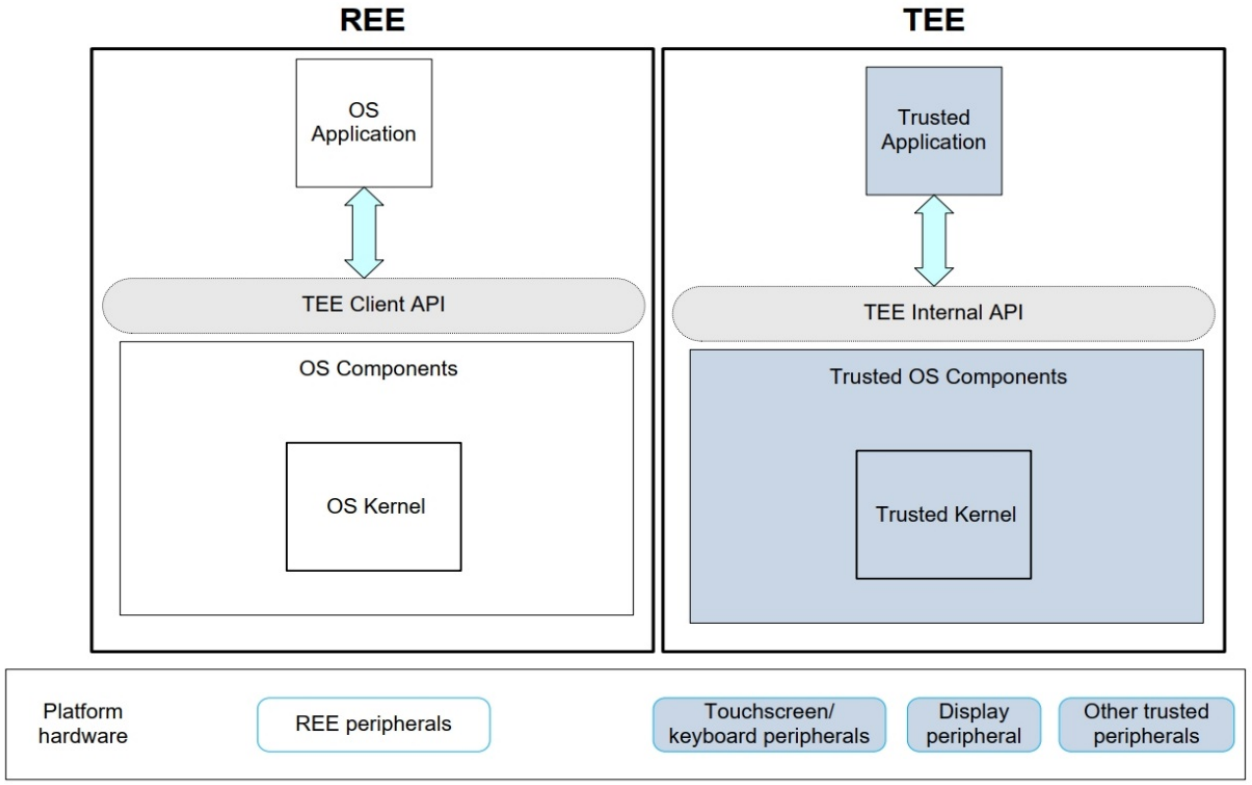


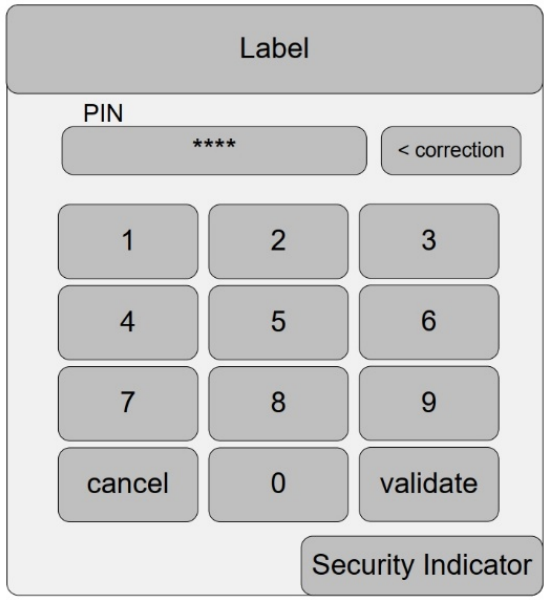
Многие современные микропроцессоры и микроконтроллеры имеют в своем составе доверенные периферийные модули, позволяющие на аппаратном уровне осуществлять поддержку ряда функций TEE API.

Доверенная среда исполнения (TEE)

Доверенный пользовательский интерфейс

Основная идея доверенного пользовательского интерфейса **(Trusted User Interface, TUI)** состоит в том, чтобы критичные операции, например, ввод пин-кода или пароля, вынести в доверенную среду исполнения и осуществлять их с помощью доверенных периферийных модулей (контроллер клавиатуры/сенсорного экрана, доверенный контроллер дисплея и т.п.).



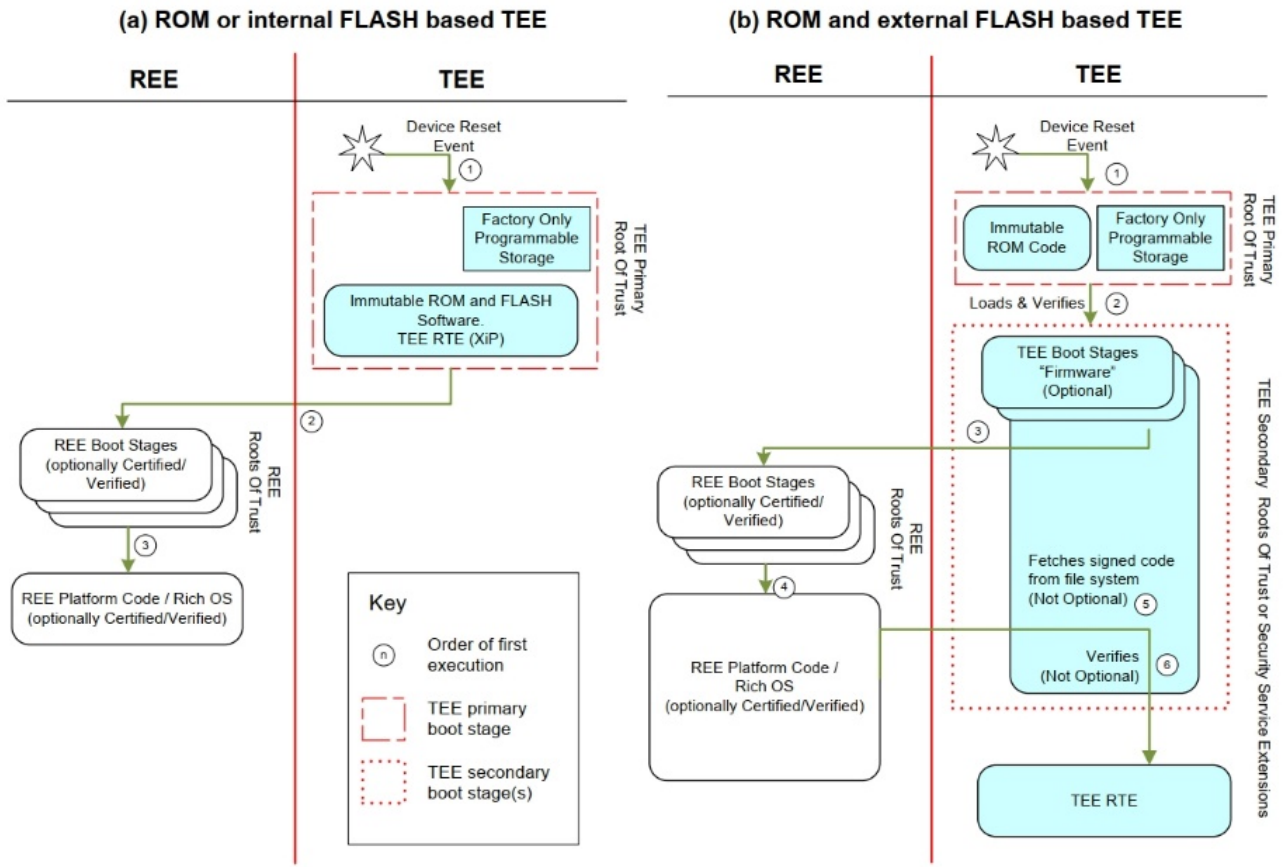


При этом возможен вывод на дисплей фрагментов пользовательского интерфейса, предназначенных для вывода (информация о платеже) или ввода (пин-код на токен) критичной информации с использованием доверенных областей памяти графического контроллера.

kaspersky

Доверенная загрузка с использованием TEE

Платформа TEE загружается из встроенного кода загрузочного масочного ПЗУ внутри СнК. Этот код ПЗУ в целом прост и достаточно функционален, чтобы позволить перейти к следующему этапу, а в хороших конструкциях выполнить минимальный POST-тест оборудования. Код ПЗУ, объединенный с вычислительным ядром и связанными с ним данными, формирует начальный компонент корня доверия (Root of Trust).



- Одноступенчатая загрузка

Одноступенчатая загрузка затем продолжит выполнение TEE RTE из накристальной flash-памяти СнК; тот же процесс можно обнаружить в SE.

- Многоступенчатая Загрузка

В качестве альтернативы код TEE в загрузочном ПЗУ может затем загружать более сложные программные компоненты этапа загрузки из энергонезависимой памяти, проверять их с помощью активов TEE (например, хэшей, хранящихся в загрузочном ПЗУ, или однократно программируемых (OTP) fuse-битов) прежде, чем окончательно выполнять их в более быстрой оперативной памяти.

kaspersky

Загрузка остальной части устройства

Во время загрузки некоторые TEE будут настраивать общие параметры изоляции хост-устройства (например, внутренние брандмауэры данных), и поэтому некоторые TEE должны загружаться до других платформ на SoC, чтобы защитить свои активы. Это «сначала загрузиться» не обязательно на платформах TEE, которые имеют свои собственные выделенные ресурсы, но даже тогда часто желательно загрузить TEE до REE, так как это позволяет TEE предоставлять услуги безопасности для REE, обеспечивая безопасную загрузку. В то время как реализации могут решить поступить иначе, TEE должен предоставлять свои услуги только после того, как REE завершит свою собственную последовательность загрузки.

Отказ в доверенной загрузке в TEE

Как правило, если во время загрузки проверка любого загруженного программного компонента завершается неудачно, обычный процесс загрузки останавливается, и устройство может перезагрузиться с возможным сообщением об ошибке/индикацией. Во многих устройствах он будет затем искать внешние flash-образы, чтобы устранить проблему, в то время как другие устройства по умолчанию используют упрощенный TEE в ПЗУ, что позволяет использовать функции аварийной службы.

Cryptographic Operations API

Algorithm Identifier	Value	Comments
TEE_ALG_AES_ECB_NOPAD	0x10000010	
TEE_ALG_AES_CBC_NOPAD	0x10000110	
TEE_ALG_AES_CTR	0x10000210	The counter SHALL be encoded as a 16-byte buffer in big-endian form. Between two consecutive blocks, the counter SHALL be incremented by 1. If it reaches the limit of all 128 bits set to 1, it SHALL wrap around to 0.
TEE_ALG_AES_CTS	0x10000310	
TEE_ALG_AES_XTS	0x10000410	
TEE_ALG_AES_CBC_MAC_NOPAD	0x30000110	
TEE_ALG_AES_CBC_MAC_PKCS5	0x30000510	
TEE_ALG_AES_CMAC	0x30000610	
TEE_ALG_AES_CCM	0x40000710	
TEE_ALG_AES_GCM	0x40000810	
TEE_ALG_DES_ECB_NOPAD	0x10000011	
TEE_ALG_DES_CBC_NOPAD	0x10000111	
TEE_ALG_DES_CBC_MAC_NOPAD	0x30000111	
TEE_ALG_DES_CBC_MAC_PKCS5	0x30000511	
TEE_ALG_DES3_ECB_NOPAD	0x10000013	Triple DES SHALL be understood as Encrypt-Decrypt-Encrypt mode with two or three keys.

Algorithm Identifier	Value	Comments
TEE_ALG_DES3_CBC_NOPAD	0x10000113	
TEE_ALG_DES3_CBC_MAC_NOPAD	0x30000113	
TEE_ALG_DES3_CBC_MAC_PKCS5	0x30000513	
TEE_ALG_RSASSA_PKCS1_V1_5_MD5	0x70001830	
TEE_ALG_RSASSA_PKCS1_V1_5_SHA1	0x70002830	
TEE_ALG_RSASSA_PKCS1_V1_5_SHA224	0x70003830	
TEE_ALG_RSASSA_PKCS1_V1_5_SHA256	0x70004830	
TEE_ALG_RSASSA_PKCS1_V1_5_SHA384	0x70005830	
TEE_ALG_RSASSA_PKCS1_V1_5_SHA512	0x70006830	
TEE_ALG_RSASSA_PKCS1_PSS_MGF1_SHA1	0x70212930	
TEE_ALG_RSASSA_PKCS1_PSS_MGF1_SHA224	0x70313930	
TEE_ALG_RSASSA_PKCS1_PSS_MGF1_SHA256	0x70414930	
TEE_ALG_RSASSA_PKCS1_PSS_MGF1_SHA384	0x70515930	
TEE_ALG_RSASSA_PKCS1_PSS_MGF1_SHA512	0x70616930	
TEE_ALG_RSAES_PKCS1_V1_5	0x60000130	
TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA1	0x60210230	
TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA224	0x60310230	
TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA256	0x60410230	
TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA384	0x60510230	
TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA512	0x60610230	
TEE_ALG_RSA_NOPAD	0x60000030	
TEE_ALG_DSA_SHA1	0x70002131	
TEE_ALG_DSA_SHA224	0x70003131	
TEE_ALG_DSA_SHA256	0x70004131	
TEE_ALG_DH_DERIVE_SHARED_SECRET	0x80000032	
TEE_ALG_MD5	0x50000001	
TEE_ALG_SHA1	0x50000002	
TEE_ALG_SHA224	0x50000003	
TEE_ALG_SHA256	0x50000004	
TEE_ALG_SHA384	0x50000005	
TEE_ALG_SHA512	0x50000006	
TEE_ALG_HMAC_MD5	0x30000001	
TEE_ALG_HMAC_SHA1	0x30000002	
TEE_ALG_HMAC_SHA224	0x30000003	

Algorithm Identifier	Value	Comments
TEE_ALG_HMAC_SHA256	0x30000004	
TEE_ALG_HMAC_SHA384	0x30000005	
TEE_ALG_HMAC_SHA512	0x30000006	
TEE_ALG_HMAC_SM3	0x30000007	If supported
TEE_ALG_ECDSA_SHA1	0x70001042	If supported
TEE_ALG_ECDSA_SHA224	0x70002042	If supported
TEE_ALG_ECDSA_SHA256	0x70003042	If supported
TEE_ALG_ECDSA_SHA384	0x70004042	If supported
TEE_ALG_ECDSA_SHA512	0x70005042	If supported
TEE_ALG_ED25519	0x70006043	If supported
TEE_ALG_ECDH_DERIVE_SHARED_SECRET	0x80000042	If supported
TEE_ALG_X25519	0x80[illegible]044	If supported
TEE_ALG_SM2_DSA_SM3	0x70006045	If supported
TEE_ALG_SM2_KEP	0x60000045	If supported
TEE_ALG_SM2_PKE	0x[illegible]045	If supported
TEE_ALG_SM3	0x50000007	If supported
TEE_ALG_SM4_ECB_NOPAD	0x10000014	If supported
TEE_ALG_SM4_CBC_NOPAD	0x10000114	If supported
TEE_ALG_SM4_CTR	0x10000214	If supported
TEE_ALG_ILLEGAL_VALUE	0xEFFFFFFF	Reserved for GlobalPlatform compliance test applications
Reserved for implementation-defined algorithm identifiers	0xF0000000 – 0xF0FFFFFF	
All other values are reserved.		

Российская криптография?

Таблица идентификаторов криптографических алгоритмов TEE Cryptographic Operations API

kaspersky

Расширения безопасности ARM TrustZone

Общие положения. Модель угроз

В состав **ARM** процессоров входят специфические расширения безопасности **ARM TrustZone**, которые на аппаратном уровне позволяют разделять ресурсы и создавать доверенные системы. Однако, если возникает необходимость их использования, то это будет накладывать определенные ограничения на операционную систему общего назначения и непривилегированный код, другими словами, код, который не является частью доверенной системы. Основной задачей **ARM TrustZone** является противодействие программным и аппаратным атакам, которые невозможно нейтрализовать с помощью обычного ПО.

Программные и аппаратные атаки можно классифицировать по следующим категориям:

Чисто программные атаки (Software attacks)
- Для атак с помощью вредоносного ПО (malicious software) обычно не требуется физического доступа к устройству, и они могут использовать уязвимости (vulnerabilities) в операционной системе и/или приложениях.

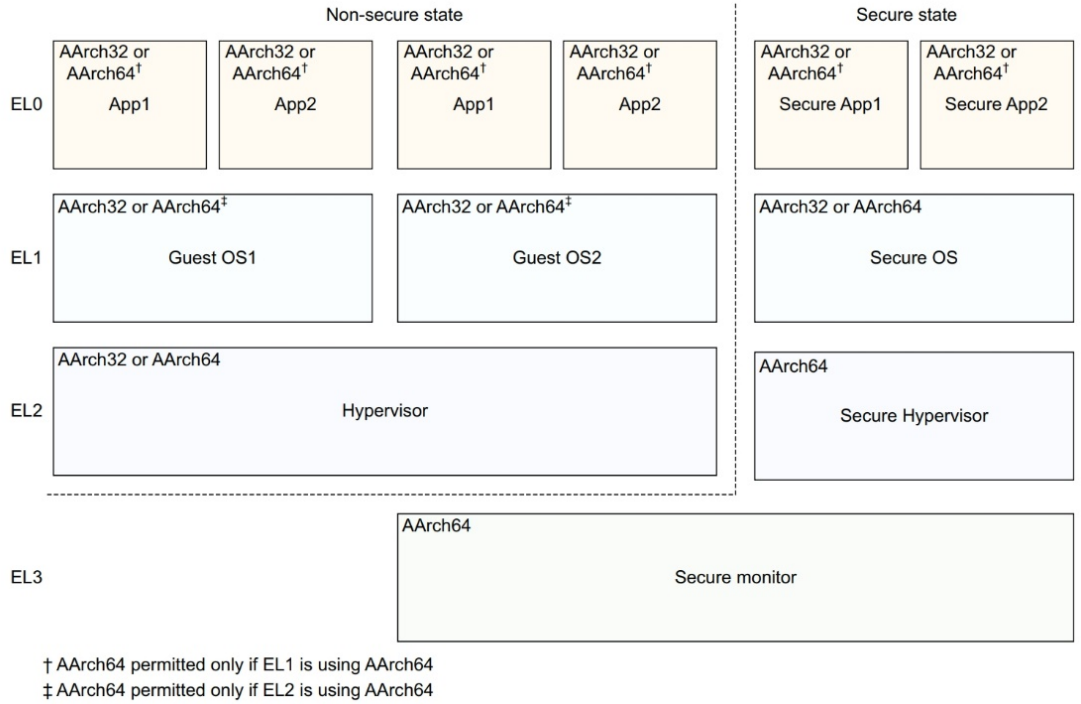
Простейшие аппаратные атаки (Simple hardware attacks)
- Они обычно являются пассивными, в большинстве случаев неразрушающими атаками, для которых требуется доступ к устройству и минимальное воздействие на аппаратную платформу, а также использование общедоступных инструментов, таких как логические пробники (анализаторы) и JTAG-отладчики.

Лабораторные аппаратные атаки (Laboratory hardware attack)
- Для проведения этого типа атак требуется сложный и дорогостоящий инструментарий, такой как технология сфокусированных ионных пучков (Focused Ion Beam, FIB), анализа утечек по воздуху и цепям питания, и такие атаки проводятся против смарткарт или аналогичных по архитектуре устройств.

Технология TrustZone разрабатывалась для противодействия **программным** и **простейшим аппаратным** атакам.

TrustZone для микропроцессоров Cortex-A

Многоуровневая архитектура ПО. Уровни исключений



Архитектура не определяет, какое программное обеспечение использует тот или иной уровень исключений. Однако имеется типовая модель использования уровней исключений:

EL0	Прикладной уровень (Applications).
EL1	Ядро ОС (OS kernel) и связанные с ним функции, которые обычно описываются как привилегированные (privileged).
EL2	Гипервизор (Hypervisor).
EL3	Защищенный режим монитора (Secure monitor).

Архитектура Armv8-A определяет набор уровней исключений (Exception levels), от EL0 до EL3, где:

- Если ELn есть уровень, то увеличивающееся значение n указывает на увеличивающийся уровень привилегий для выполняющегося ПО.
- Выполнение ПО на уровне EL0 называется непривилегированным выполнением (unprivileged execution).
- Уровень EL2 обеспечивает поддержку для виртуализации.
- Уровень EL3 обеспечивает поддержку переключения между двумя состояниями безопасности (Security states): защищенным (Secure state) и незащищенным (Non-secure state).

Реализация может содержать не все уровни исключений. Обязательным является наличие EL0 и EL1. Уровни EL2 и EL3 являются опциональными.

kaspersky

Состояния защиты (Security state)

В общем случае архитектура Armv8-A предусматривает два состояния защиты (Security states), каждое с индивидуальным адресным пространством физической памяти.
Разделение общего адресного пространства на защищенное и незащищенное происходит с помощью дополнительной адресной линии, которая соответствует старшему биту адреса, или NS-биту.
Управление данным битом происходит в блоке управления памятью MMU.

Защищенное состояние **Secure state**	При нахождении в этом состоянии, процессорный элемент может иметь доступ как к защищенному физическому адресному пространству (Secure physical address space), так и к незащищенному физическому адресному пространству (Non-secure physical address space).
Незащищенное состояние **Non-secure state**	При нахождении в этом состоянии процессорный элемент: • Может иметь доступ только к незащищенному физическому адресному пространству (Non-secure physical address space). • Не может иметь доступ к защищенным управляющим системным ресурсам (Secure system control resources).

Модель защиты в Armv8-A

Если реализация включает уровень EL3, то имеются 2 состояния защиты (Security states), защищенное (Secure state) и незащищенное (Non-secure), и:

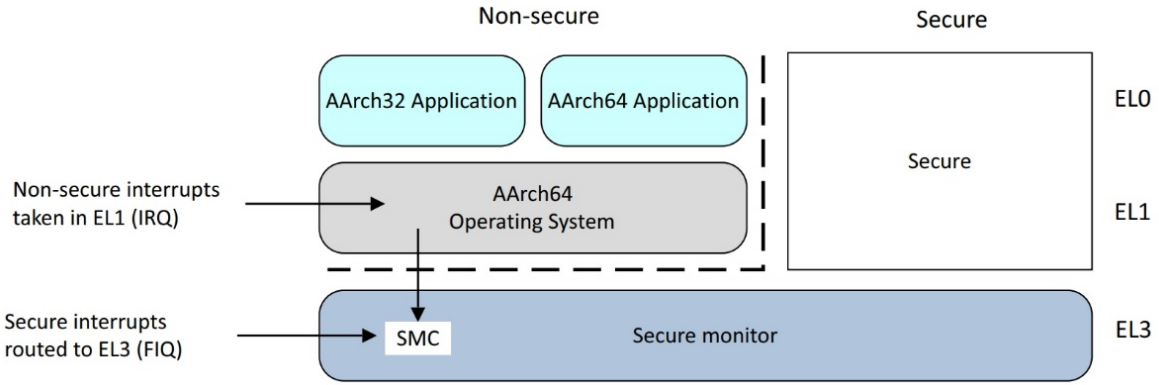
- Уровень EL3 существует только в защищенном состоянии (Secure state)
- Переход из незащищенного состояния (Non-secure state) в защищенное состояние (Secure state) может произойти только при возникновении исключения к EL3
- Переход из защищенного состояния (Secure state) в незащищенное состояние (Non-secure state) может произойти только при возврате исключения из EL3
- Если расширения ARMv8.4-SecEL2 не реализованы, то EL2 существует только в незащищенном состоянии (Non-secure state)
- Если расширения ARMv8.4-SecEL2 реализованы, то EL2 может существовать и в защищенном состоянии (Secure state). Активируется при SCR_EL3.EEL2 = 1

Если реализация не включает EL3, то имеется только защищенное состояние (Security state), которое:

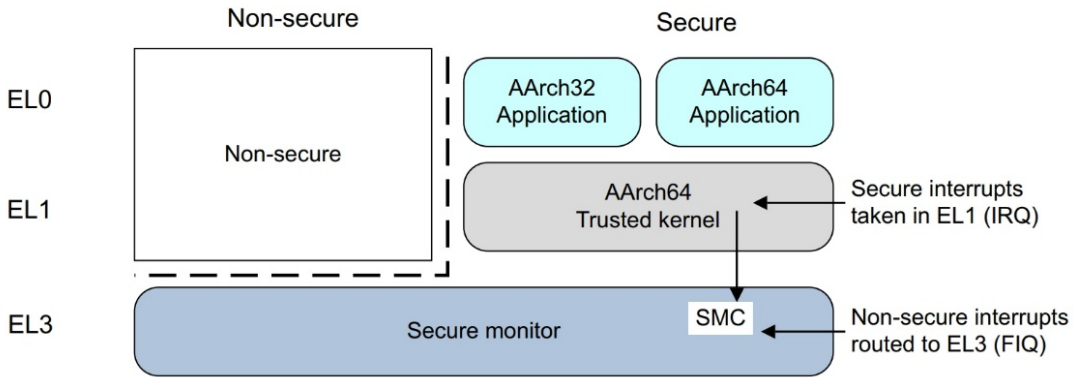
- ЗАВИСИТ ОТ РЕАЛИЗАЦИИ, если реализация не имеет EL2 или если реализованы расширения ARMv8.4-SecEL2
- Являетеся незащищенным состоянием (Non-secure state), если реализация включает EL2 и не реализованы расширения ARMv8.4-SecEL2

Переключение между состояниями защиты. Прерывания

Поскольку ядра выполняют код из двух миров, переключение контекста между ними происходит через выполнение инструкции Secure Monitor Call (SMC) или с помощью аппаратных механизмов исключения, таких как прерывания. Процессоры ARM имеют два типа прерываний: FIQ и IRQ.



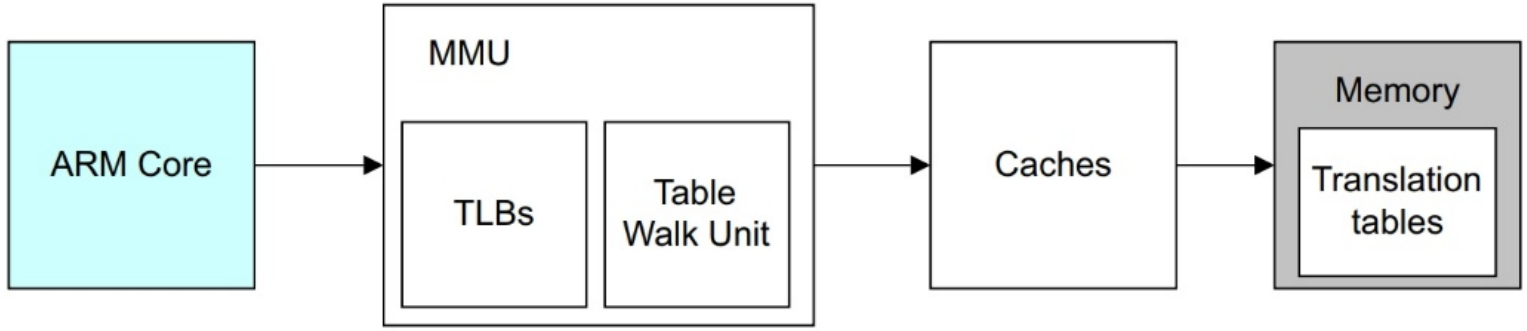
Незащищенные прерывания



Защищенные прерывания

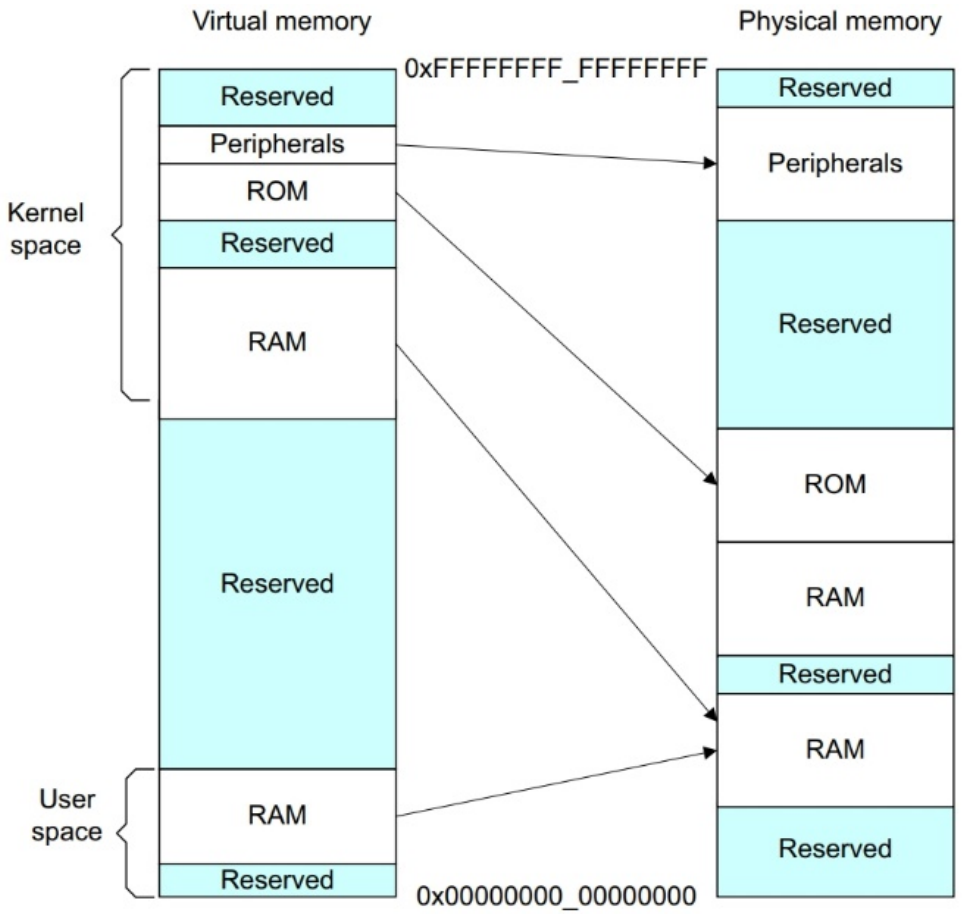
Существует явная поддержка безопасных прерываний в виде элементов управления для перенаправления исключений и прерываний на EL3, независимо от текущего DAIF. Однако эти элементы управления различают только основные типы прерываний: IRQ, FIQ и асинхронные прерывания. Более мелкозернистый контроль требует, чтобы прерывания фильтровались в безопасные и небезопасные группы. Чтобы сделать это эффективно, требуется поддержка со стороны контроллера прерываний GIC, который имеет для этого явные возможности. Типичный пример использования заключается в том, что FIQ используется в качестве защищенных прерываний, отображая защищенные источники прерываний как FIG в контроллере прерываний. Соответствующие регистры периферийных устройств и контроллеров прерываний должны быть помечены только как защищенные, чтобы предотвратить повторную настройку этих прерываний в обычном мире.

Управление памятью. Memory Management Unit (MMU)



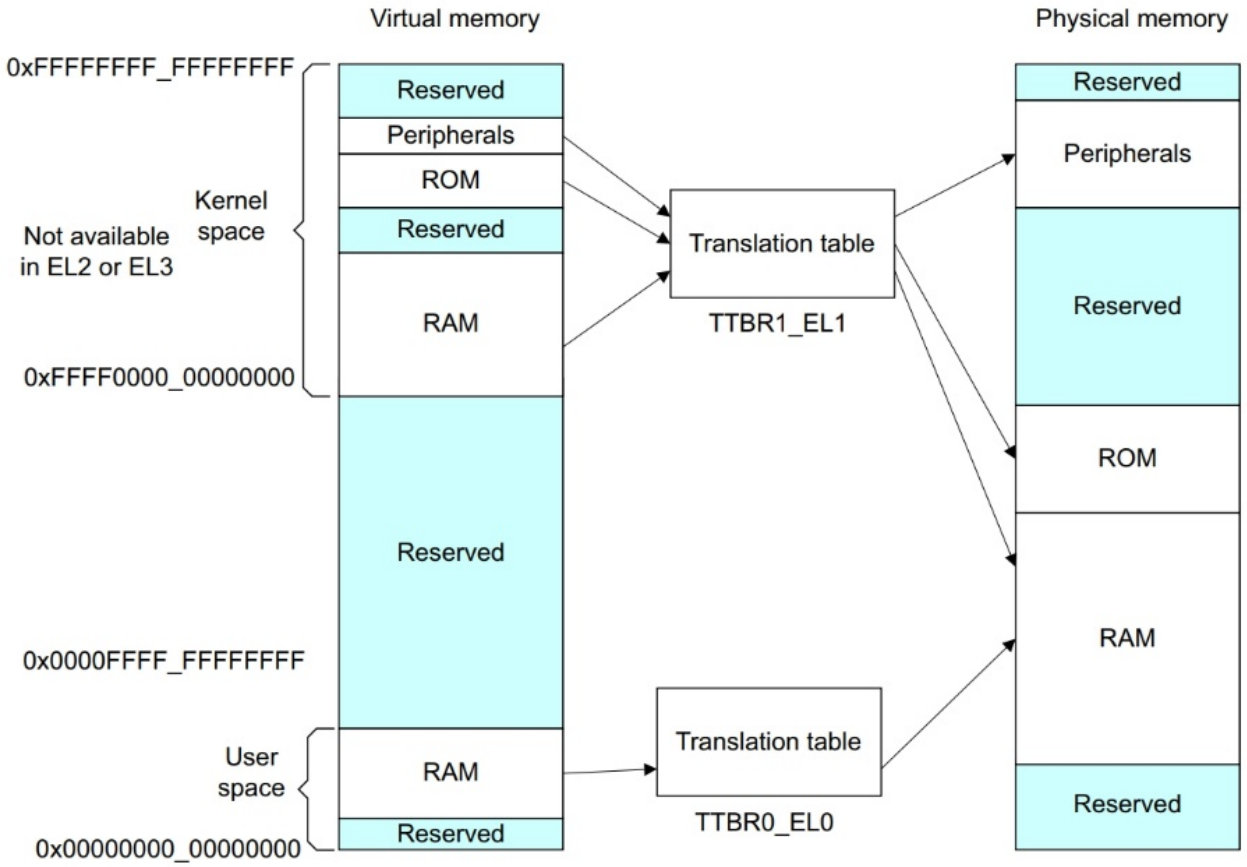
Важной функцией блока управления памятью (MMU) является обеспечение возможности выполнения Системой нескольких задач в виде независимых программ, работающих в своем собственном виртуальном пространстве памяти. Он не нуждается в каких- либо знаниях о физической карте памяти системы, то есть об адресах, которые фактически используются аппаратным обеспечением , или о других программах, которые могут выполняться одновременно.
Возможно использовать одно и то же адресное пространство виртуальной памяти для каждой программы. Также можно работать с непрерывной картой виртуальной памяти, даже если физическая память фрагментирована. Это виртуальное адресное пространство отделено от реальной физической карты памяти в системе. Возможно писать, компилировать и связывать приложения для запуска в виртуальном пространстве памяти.

Управление памятью. Memory Management Unit (MMU)



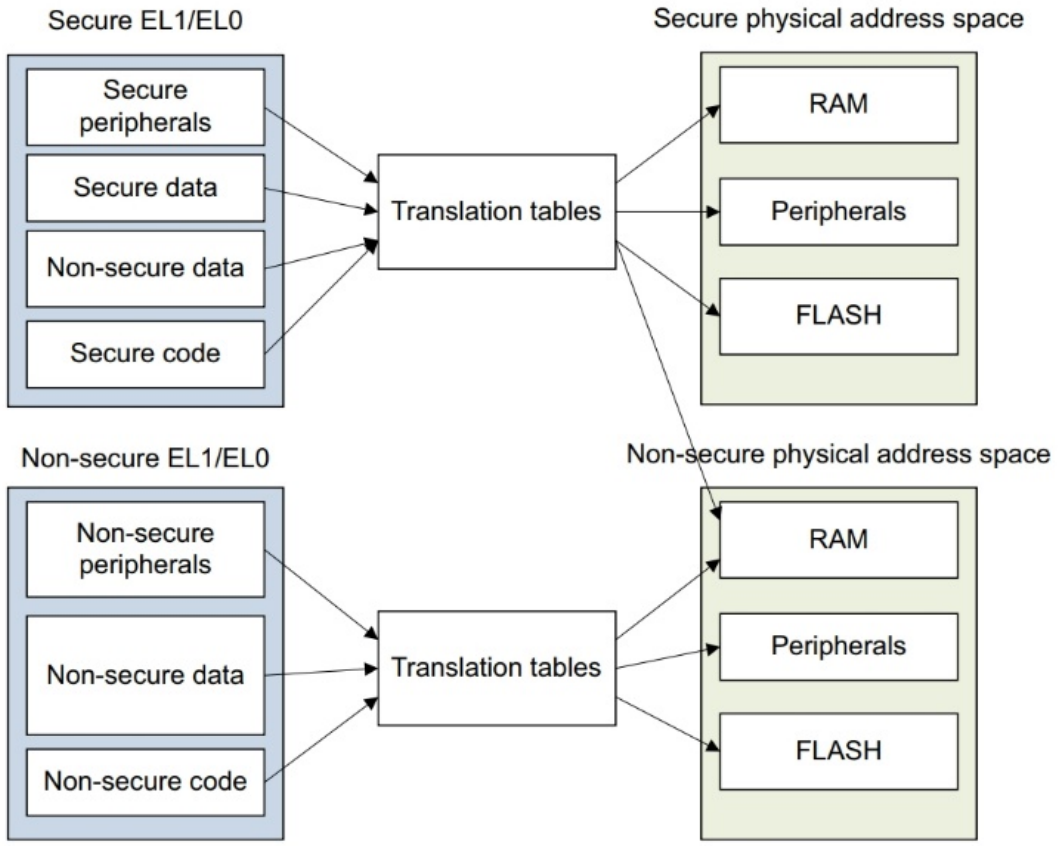
Различные процессоры и устройства в одной системе могут иметь различные виртуальные и физические карты адресов. ОС программирует MMU для перевода между этими двумя видами памяти.
Для этого аппаратное обеспечение в системе виртуальной памяти должно обеспечивать преобразование адресов, которое представляет собой преобразование виртуального адреса, выданного процессором, в физический адрес в основной памяти.
Виртуальные адреса – это те, которые используются вами, компилятором и компоновщиком при размещении кода в памяти.
Физические адреса – это те, которые используются реальной аппаратной системой.
MMU использует старшие биты виртуального адреса для индексирования записей в таблице перевода (TT) и установления того, к какому блоку осуществляется доступ. MMU преобразует виртуальные адреса кода и данных в физические адреса в реальной системе. Перевод осуществляется автоматически в аппаратном обеспечении и является прозрачным для приложения. Помимо преобразования адресов, MMU управляет разрешениями доступа к памяти, упорядочением памяти и политиками кэша для каждой области памяти.

Управление памятью. Memory Management Unit (MMU)



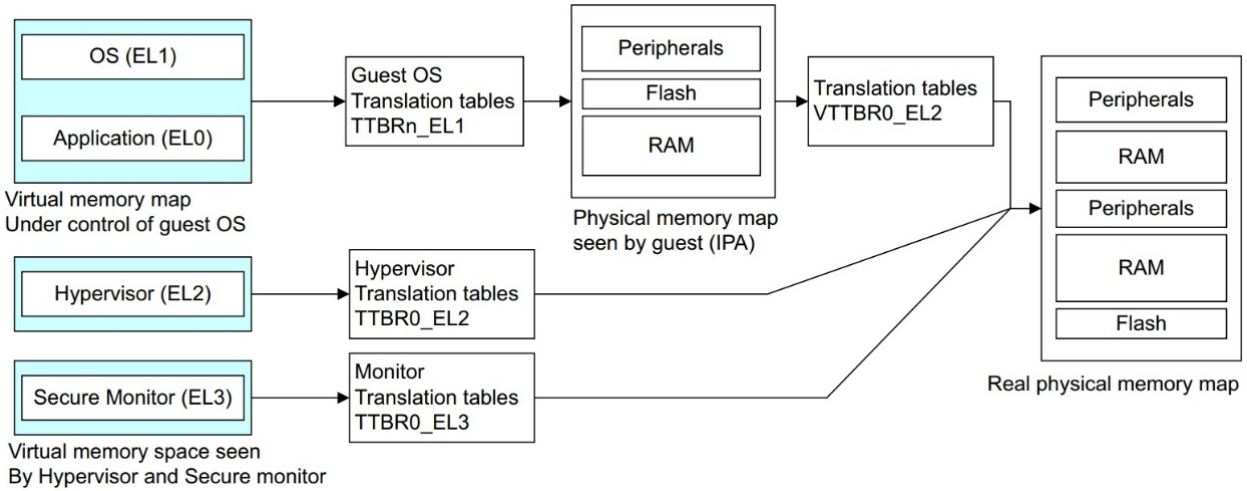
MMU позволяет записывать задачи или приложения таким образом, чтобы они не имели никаких знаний о физической карте памяти системы или о других программах, которые могут выполняться одновременно. Это позволяет использовать одно и то же адресное пространство виртуальной памяти для каждой программы. Он также позволяет работать с непрерывной картой виртуальной памяти, даже если физическая память фрагментирована. Это виртуальное адресное пространство отделено от реальной физической карты памяти в системе. Приложения записываются, компилируются и связываются для запуска в виртуальном пространстве памяти.

Управление памятью. Защищенное и незащищенное адресное пространство



Теоретически, защищенные и незащищенные физические адресные пространства независимы друг от друга и существуют параллельно. Система может быть спроектирована так, чтобы иметь две совершенно разные системы памяти. Однако большинство реальных систем рассматривают защищенность и незащищенность как атрибут контроля доступа. Обычная (незащищенная) среда исполнения (Normal World) может получить доступ только к незащищенному физическому адресному пространству. Защищенная среда исполнения (Secure World) может получить доступ к обоим физическим адресным пространствам. Это контролируется с помощью таблиц перевода.

Управление памятью. Трансляция на уровнях EL2 и EL3



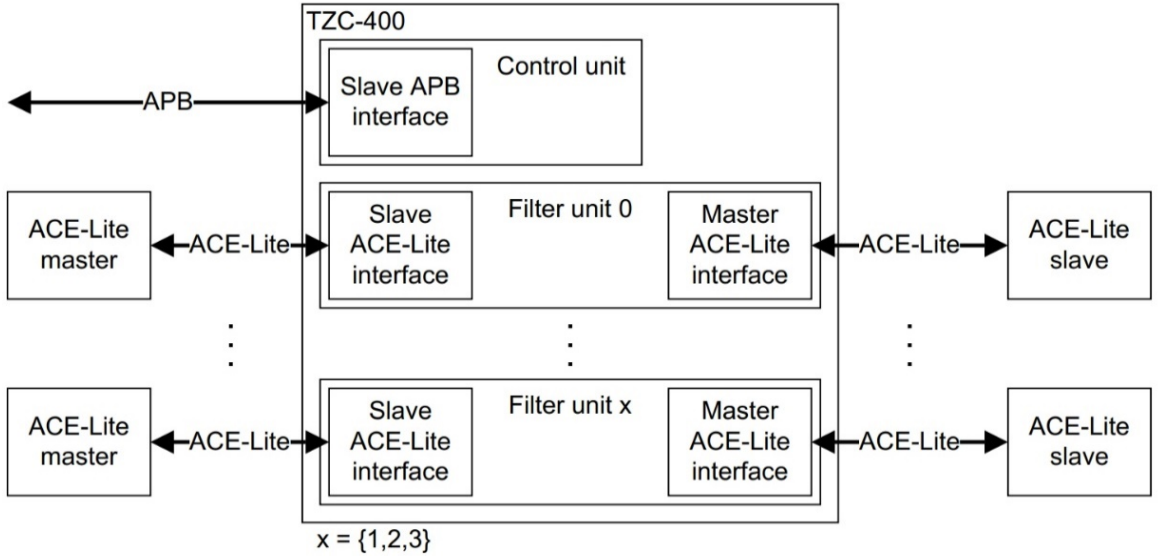
Расширения виртуализации для архитектуры ARMv8-A вводят второй этап трансляции. При наличии гипервизора в системе может присутствовать одна или несколько гостевых операционных систем. Они продолжают использовать **TTBRn_EL1**, и использование MMU продолжается в неизменном виде. Гипервизор должен выполнить несколько дополнительных шагов преобразования в двухэтапном процессе для совместного использования физической системы памяти между различными гостевыми операционными системами.

На первом этапе виртуальный адрес (VA) переводится в промежуточный физический адрес (IPA). Обычно это происходит под контролем ОС. На втором этапе, управляемом гипервизором, выполняется перевод IPA в конечный физический адрес (PA). Гипервизор и защищенный монитор также имеют свой набор таблиц перевода этапа 1 для их собственного кода и данных, которые выполняют отображение непосредственно из VA в PA.

Защищенный монитор на EL3 имеет свои собственные таблицы перевода, которые способны получать доступ как к защищенным, так и к незащищенным физическим адресам. **TTBR0_EL3** используется только в режиме **Secure monitor EL3**, а не самим доверенным ядром. Когда переход к защищенной области (Secure World) завершен, доверенное ядро использует переводы EL1. Поскольку эти регистры не хранятся в банке данных **AArch64**, код **Secure monitor** должен сконфигурировать новые таблицы для Secure World и сохранять и восстанавливать копии **TTBR0_EL1** и **TTBR1_EL1**.

kaspersky

Контроллер TrustZone TZC-400



TZC-400 выполняет проверку безопасности транзакций с памятью или периферийными устройствами. Его можно использовать для создания до восьми отдельных областей в адресном пространстве, каждая из которых имеет индивидуальную настройку уровня безопасности. Любые транзакции должны соответствовать требованиям безопасности, чтобы получить доступ к памяти или периферийному устройству. Возможно запрограммировать базовый адрес, верхний адрес, разрешение использования и параметры безопасности для каждого региона.

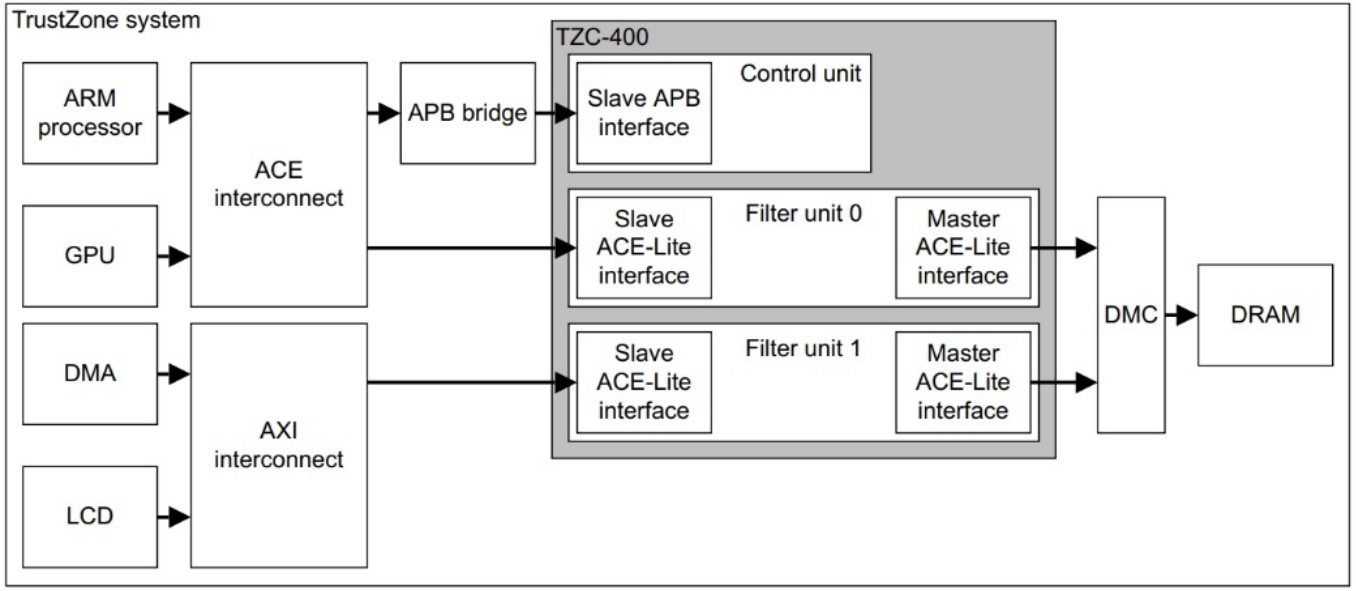
TZC-400 работает между ведущими (master) и ведомыми (slave) устройствами ACE-Lite в системе TrustZone и фильтрует доступ по шине от ведущих к ведомым устройствам. Он выполняет фильтрацию на основе требований безопасности, которые указаны для адресных областей. Кроме конфигурирования восьми адресных областей TZC-400 можно запрограммировать для сообщения о неисправностях с помощью канала ответа ACE-Lite или прерываний.

Фильтрующие блоки выполняют проверку безопасности. Каждый блок фильтра имеет ведомый и ведущий интерфейс ACE-Lite . Все блоки фильтра работают от одного набора общих регистров конфигурации региона. Это обеспечивает согласованность во всех блоках фильтров. Входы Fast Path IDentity (FPID) указывают блокам фильтров, что они должны обрабатывать доступ с меньшей задержкой, чем для обычного доступа по пути.

Каждый блок фильтра работает в своем собственном тактовом домене и имеет интерфейс AXI low-power для использования в вашей стратегии стробирования часов. Каждый домен может работать асинхронно со всеми другими доменами. Интерфейс APB блока управления позволяет запрограммировать параметры безопасности и адреса каждого региона. Блок управления также имеет маломощный интерфейс AXI для использования в вашей стратегии стробирования часов.

kaspersky

Пример системы с использованием контроллера TZC-400



В примере имеется два блока фильтров, потому что есть два пути из ACE-Lite в контроллер динамической памяти (DMC).

Эти два пути таковы:

- Один из них от моста AXI interconnect к ведущим AXI masters контроллеров DMA и LCD.
- Другой от моста ACE interconnect к ведущим ACE и ACE-Lite masters процессора и графического процессора (GPU).

Оба ведущих ACE-Lite интерфейса подключены к DMC. Это обеспечивает полный контроль доступа к DRAM.

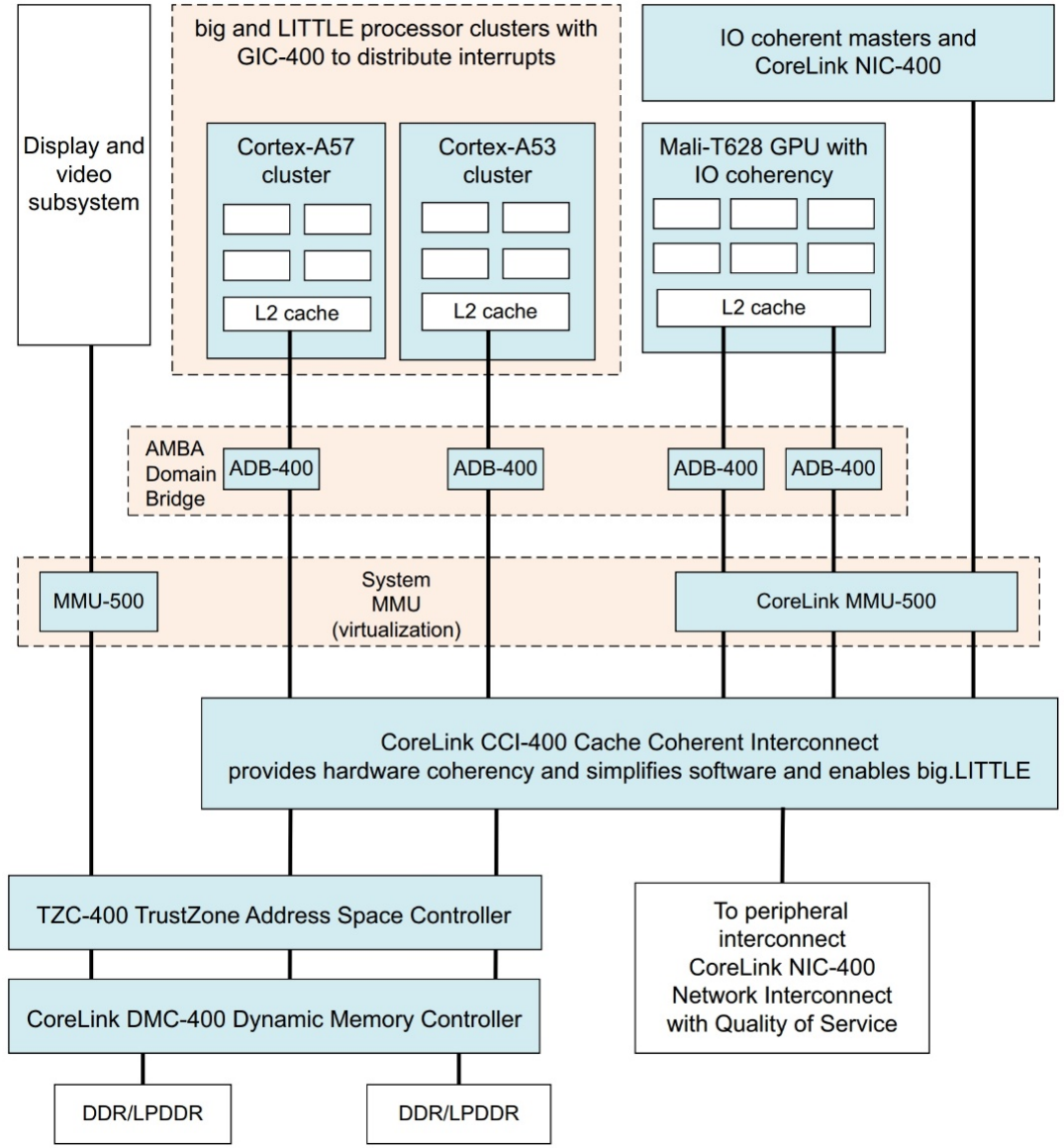
Доверенная отладка

Система безопасности контролирует также доступ к отладочному интерфейсу аппаратного обеспечения. Имеется возможность настроить полный JTAG-интерфейс для управления доступом к обычной (**Normal**) и защищенной (**Secure**) программной среде исполнения таким образом, что не будет никакой утечки информации из доверенной среды исполнения.

Настройками аппаратного обеспечения можно управлять через защищенное периферийное устройство, а также через выводы микропроцессора с помощью следующих сигналов:

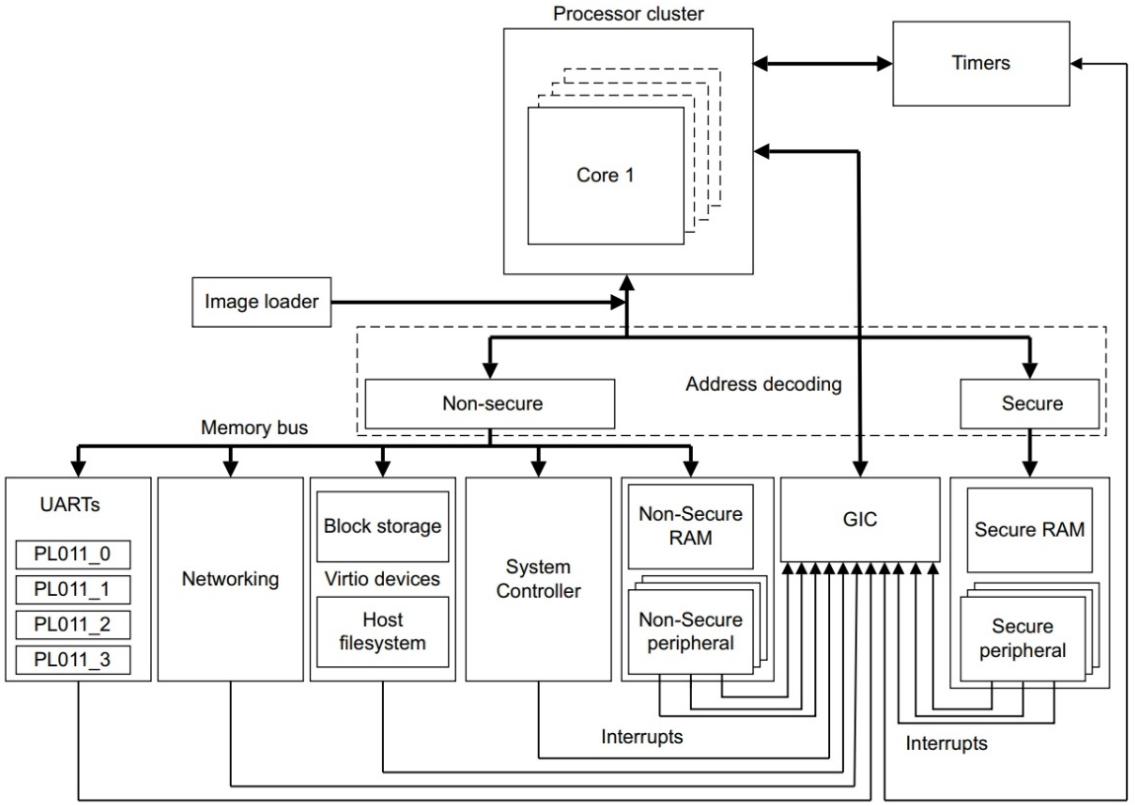
- **Secure Privileged Invasive Debug Enable (SPIDEN)**: активация доверенной JTAG-отладки.
- **Secure Privileged Non-Invasive Debug Enable (SPNIDEN)**: активация монитора трассировки и производительности

Пример архитектуры микроконтроллера ARMv8-A



kaspersky

Аппаратная платформа ARMv8-A Foundation Platform

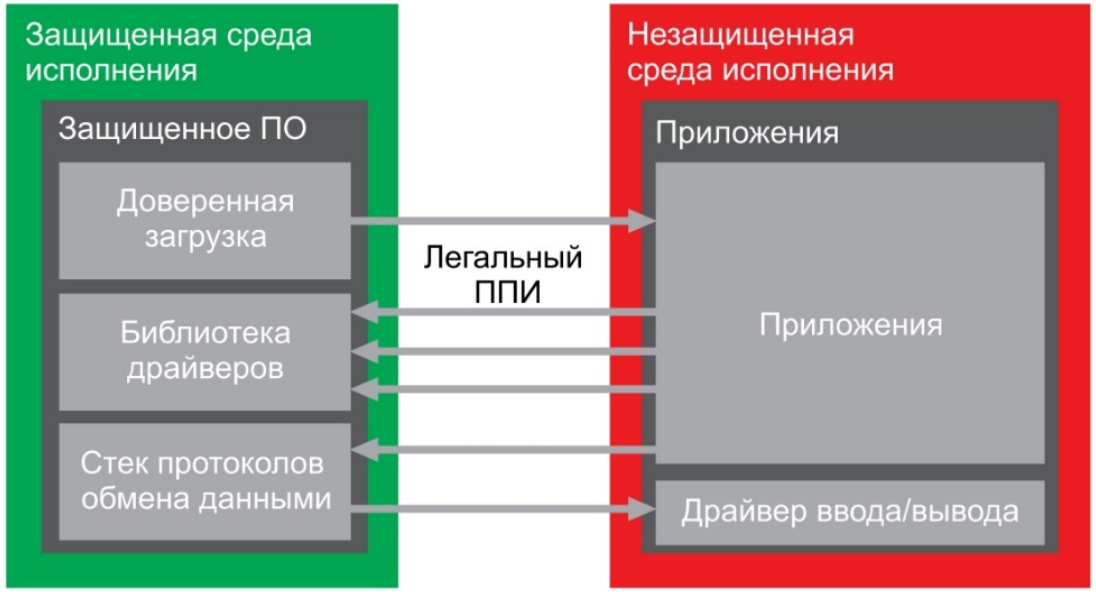


В состав **Foundation Platform** входят:

- Процессорный кластер **ARMv8-A**, содержащий **1-4** ядра, в котором реализованы:
  - Все уровни исключений **AArch64**
  - Поддержка **AArch32** для **EL0** и **EL1**
  - **Little** и **big endian** на всех уровнях исключений
  - Универсальные таймеры
  - **Self-hosted** отладка
  - Отображенные на адресное пространство **GICv2** и **GICv3**
- **8GB** ОЗУ.
- Четыре **PL011 UART,** подключенные как **xterms**.
- Периферийные модули: **Real-time clock**, **Watchdog timer**, **Real-time timer** и **Power controller**.
- Доверенные периферийные модули: Trusted watchdog, Random number generator, Non-volatile counters и Root-key storage.
- Сетевое устройство, подключаемое к сетевым ресурсам хоста.
- Блочное запоминающее устройство, реализованное в виде файла на хосте.
- Небольшой блок системного регистра со светодиодом и переключателями, видимыми с помощью веб-сервера.
  - Простейший веб-интерфейс, отображающий состояние модели.
  - Доступ к файловой системе хоста реализован как файловая система **Plan 9 Filesystem**.

TrustZone для микроконтроллеров Cortex-M

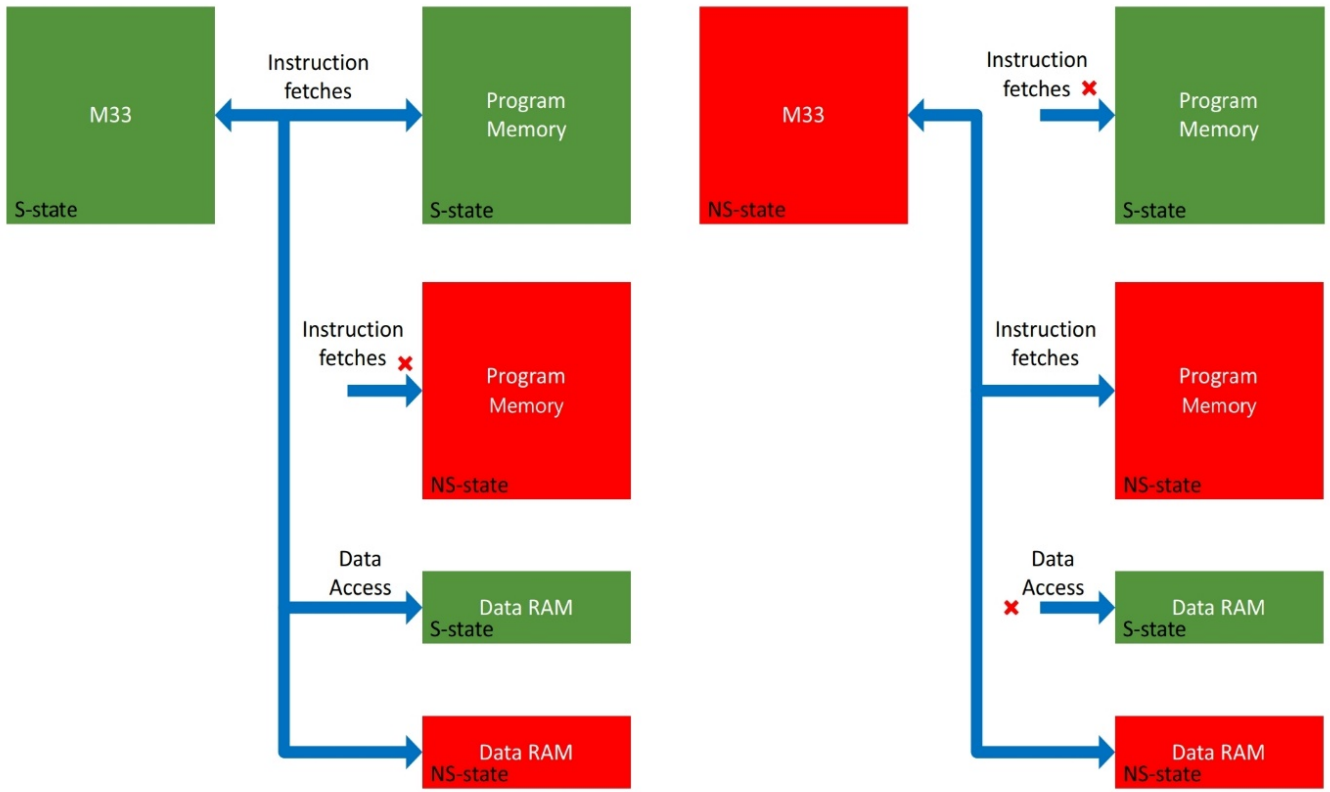
Защита продуктов для IoT



Технология TrustZone пригодна для реализации дополнительных функций защиты (**Protection features**) в расширенных микроконтроллерах, предназначенных для интернета вещей (**internet of things, IoT**). Например, если в спроектированном для IoT-приложений микроконтроллере содержится ряд функций защиты (**Security features**), то применение технологии TrustZone даст уверенность в том, что все они будут использованы только с помощью специальных вызовов API с правильными точками входа.
При использовании технологии TrustZone для защиты этих функций, разработчик может:

- запретить недоверенным приложениям (**Untrusted applications**) непосредственный доступ к критичным ресурсам;
- быть уверенным в том, что образ flash-памяти можно будет запрограммировать только после проверки его подлинности;
- защитить встроенное ПО от попыток инженерного анализа;
- хранить секретную информацию с защитой на уровне ПО.

Защищенное и незащищенное состояние



При включенной **TrustZone** для микроконтроллера **Cortex-M33** (**CM33**) справедливы следующие правила:

- Защищенное состояние **CPU CM33** (**CPU-S**) может выполнять инструкции только из защищенной памяти (**S-memory**).
- Состояние **CPU-S** может иметь доступ на чтение и на запись к данным как в защищенной памяти (**S-memory**), так и в незащищенной памяти (**NS-memory**).
- Незащищенное состояние **CPU CM33** (**CPU-NS**) может выполнять инструкции только из незащищенной памяти (**NS-memory**).
- Состояние **CPU-NS** может иметь доступ к данным только в незащищенной памяти (**NS-memory**).

kaspersky

Защищенное и незащищенное состояние

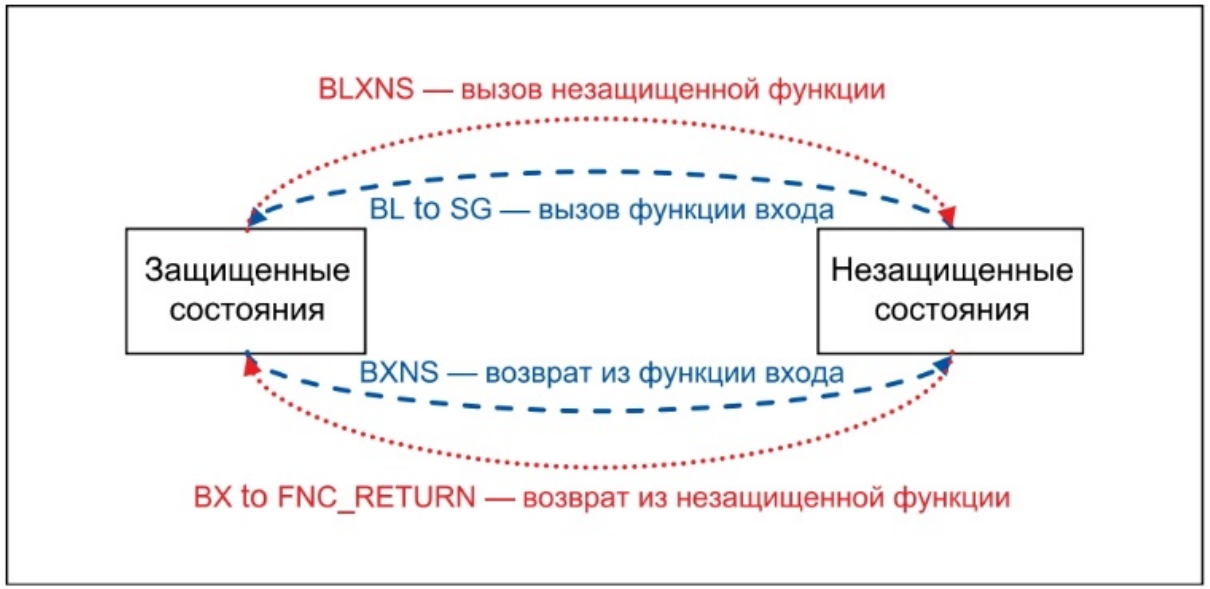
Иными словами:

- Код незащищенного приложения (**NS application code**) верит, что код защищенного приложения (**S application code**) не будет, непреднамеренно или намеренно, пытаться повредить или модифицировать незащищенный код (**NS code**) или незащищенные данные (**NS data**), чтобы вызвать неисправность или создать угрозу безопасности
- Код защищенного приложения (**S application code**) не доверяет коду незащищенного приложения (**NS application code**) и запрещает доступ из CPU-NS

Для поддержки защищенного состояния (**Secure state**), архитектура Cortex-M33 расширена рядом аппаратных блоков, таких как:

- Защищенный блок защиты памяти (**Secure MPU**)
- Защищенный контроллер прерываний (**Secure NVIC**)
- Защищенный системный таймер (**Secure SysTick**)
- Защищенный указатель стека (**Secure Stack Pointer**) с блоком проверки границы стека (**Stack-Threshold Check**)

Переход между состояниями защиты. Инструкция SG

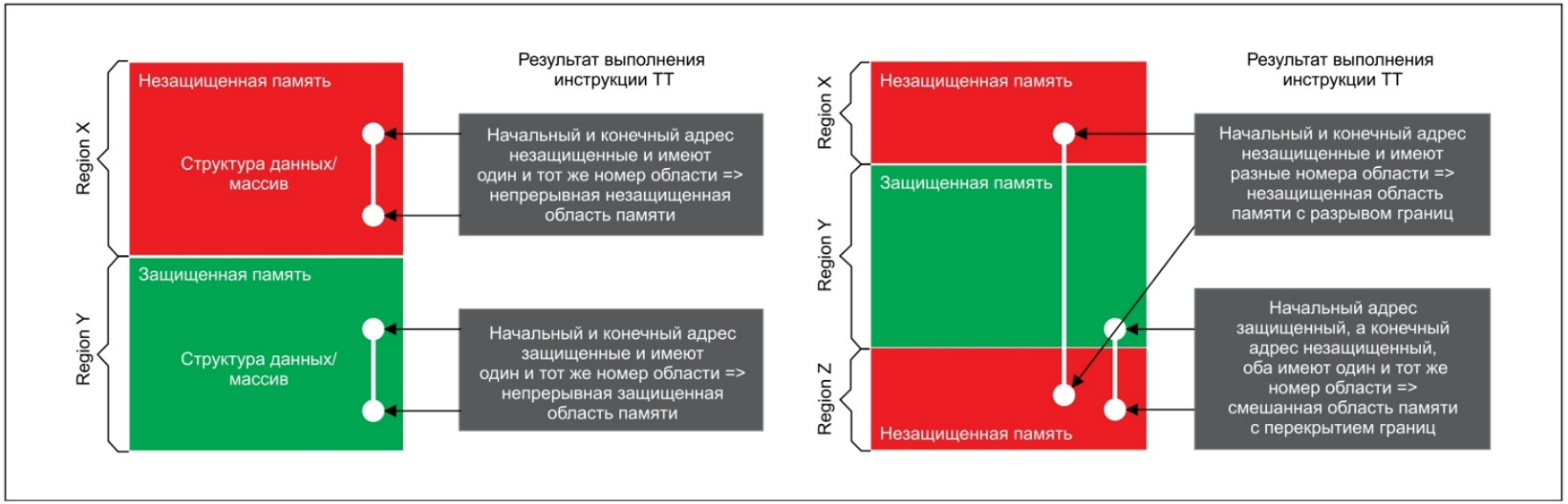


Система загружается в защищенном состоянии и может изменять состояния защиты, используя ветвления (переходы).
Переход из защищенного состояния в незащищенное может быть инициализирован программным обеспечением с помощью инструкций BXNS и BLXNS, которые имеют неустановленным младший бит (LSB).
Переходы из незащищенного в защищенное состояние могут инициализироваться программным обеспечением двумя путями:

- переход по защищенному шлюзу
- переход по зарезервированному значению FNC_RETURN

Защищенный шлюз всегда возникает по инструкции SG, которая может располагаться только в NSC области. При переходе на защищенный шлюз из незащищенного состояния инструкция SG переключает состояние защиты и очищает младший бит (LSB) адреса возврата (**return address**) в регистре LR (**Link Register**).
В любой другой ситуации инструкция SG не изменяет состояние защиты и адрес возврата.
Переход на зарезервированное значение FNC_RETURN приводит к тому, что аппаратное обеспечение осуществляет переход в защищенное состояние, снимает адрес с верхушки защищенного стека и осуществляет передачу управления по этому адресу. Зарезервированное значение FNC_RETURN записывается в LR при выполнении инструкции BLXNS.
Переход между состояниями защиты может выполняться аппаратным обеспечением при обработке прерываний. Эти переходы являются прозрачными для ПО.

Инструкция TT



В архитектуре ARMv8-M вводится новая инструкция **Test Target** (TT), которая принимает адрес памяти и возвращает конфигурацию, записанную для этого адреса в блоке защиты памяти (**Memory Protection Unit, MPU**). Дополнительный флаг T управляет тем, будут ли возвращены права доступа (**permissions**) для привилегированного (**privileged**) или непривилегированного (**unprivileged**) режима исполнения команд (**execution mode**).
При выполнении инструкции TT в защищенном состоянии результат расширяется и возвращает конфигурацию не только SAU, но и IDAU по заданному адресу. Блок MPU разделен на банки между двумя состояниями защиты.
Опциональный флаг A позволяет инструкции TT читать значения, записанные в MPU для незащищенного состояния, когда инструкция TT исполняется в защищенном состоянии.
Инструкция TT используется для проверки разрешений доступа (**access permissions**), с помощью которых различаются защищенные состояния и привилегированные уровни (**privilege levels**) для областей памяти по заданному адресу.

kaspersky

Распределение памяти

В архитектуре защитных расширений реализовано адресное пространство размером 4 Гбайт (ограничение 32-разрядной архитектуры), разделенное на защищенные (**Secure memory regions**) и незащищенные (**Non-secure memory regions**) области памяти. Область защищенной памяти (**Secure memory space**), в свою очередь, делится на области памяти двух типов: защищенную (**Secure, S**) и доступную для незащищенных обращений (**Non-secure Callable, NSC**).

Защищенное адресное пространство (**Secure addresses**) используется для областей памяти и периферийных модулей (**Peripherals**), которые должны быть доступны только из защищенного ПО или с помощью защищенных ведущих (**Secure masters**).

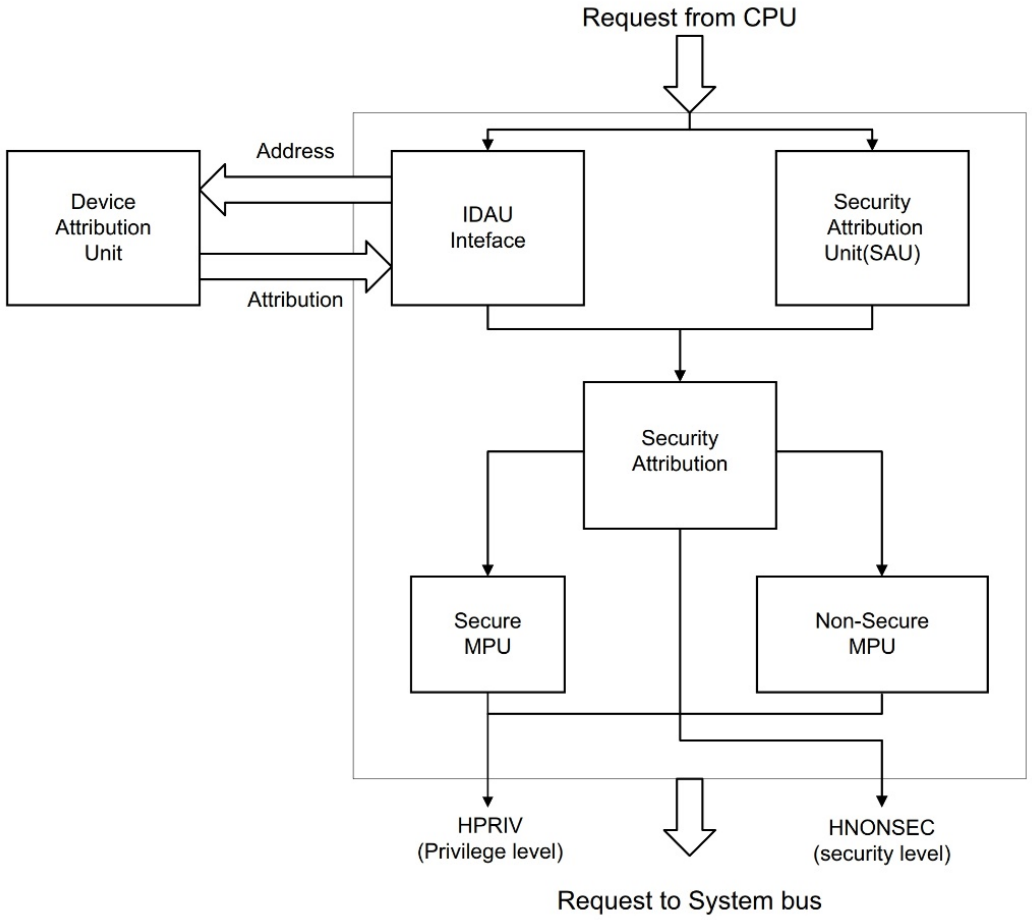
Память NSC представляет собой специальный тип защищенных областей (**Secure location**). Этот тип памяти является единственным, в котором процессор ARMv8-M допускает хранение инструкции SG, которая позволяет программным способом переходить из незащищенного состояния в защищенное. Введение областей памяти NSC избавляет создателей защищенного ПО от необходимости проверки случайного размещения инструкций SG или значений кода операции инструкции SG в обычной защищенной памяти путем ограничения области действия инструкции SG только на NSC-память.

Обычно области памяти NSC содержат таблицы виниров коротких переходов (**Short branch veneers**) или точек входа. Чтобы предотвратить ветвление незащищенных приложений в недопустимые точки входа, существует инструкция **Secure Gateway** (SG).

Когда незащищенная программа (**Non-secure program**) вызывает функцию на защищенной стороне (**Secure side**), то:

- первой инструкцией в API должна быть инструкция SG;
- инструкция SG должна располагаться в области NSC, которая определяется с помощью блока атрибутов защиты (**Security Attribution Unit, SAU**) или зависящего от реализации блока атрибутов (**Implementation Defined Attribution Unit, IDAU**).

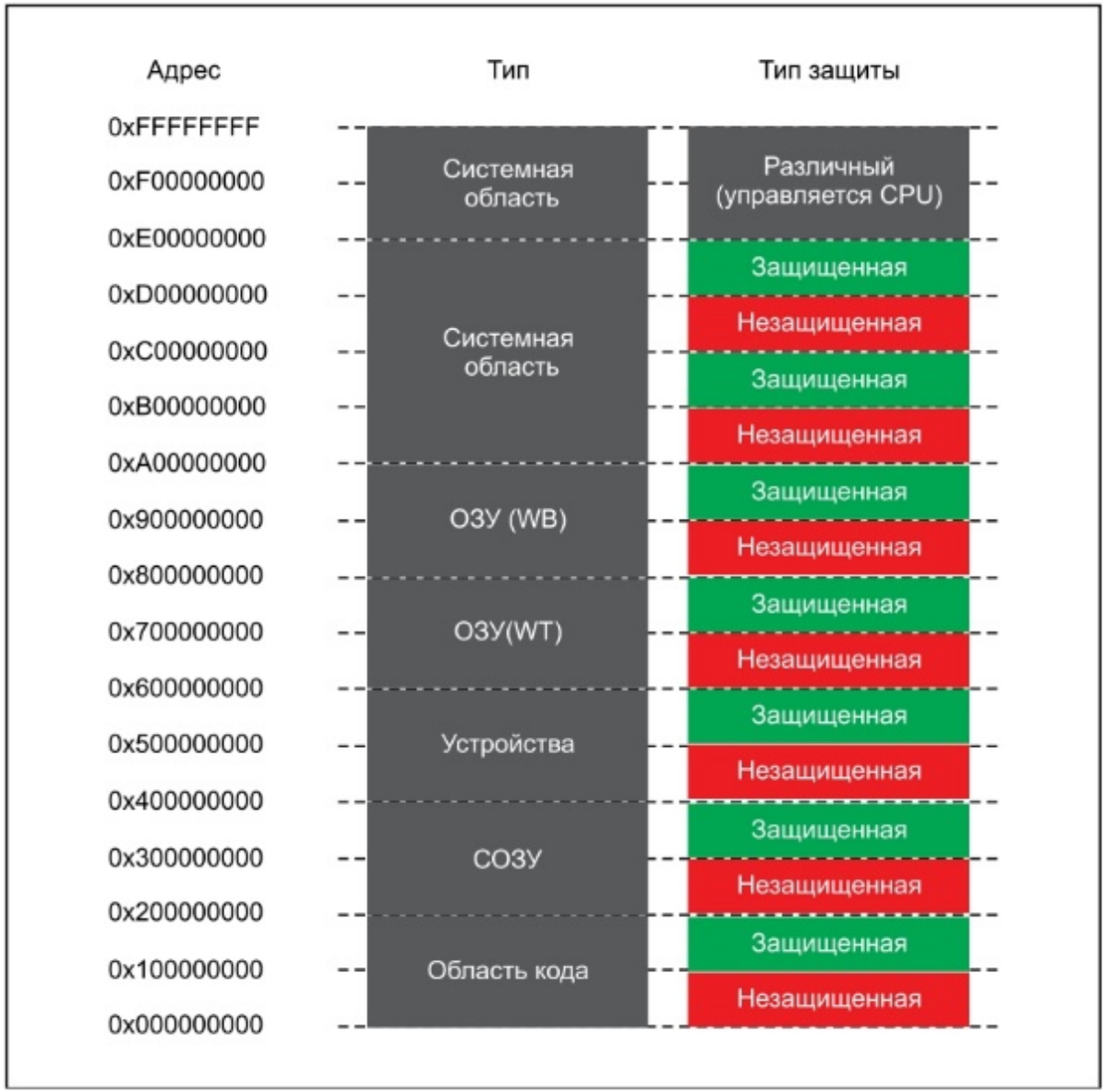
Блоки атрибутов защиты



Если в состав процессора **ARMv8-M** входят расширения безопасности, то состояние защиты области памяти контролируется комбинацией внутреннего блока атрибутов защиты (**Secure Attribution Unit, SAU**) или внешнего зависящего от реализации блока атрибутов (**Implementation Defined Attribution Unit, IDAU**). Количество областей **SAU** задается при проектировании процессора. После сброса блок **SAU** отключен.
Если не определено ни одной области **SAU** или блок **SAU** отключен и блок **IDAU** не включен в состав системы, то все адресное пространство памяти (**Memory address space**) задается как защищенное, и процессор не может переключиться в незащищенное состояние. Любые попытки переключения в незащищенное состояние приведут к ошибке. Это состояние является для процессора состоянием по умолчанию.
Блоки **SAU** и **IDAU** задают также количество подобластей для каждой области памяти. Количество областей задается 8-битным числом и используется инструкцией **Test Target** (**TT**), которая позволяет программному обеспечению определить права доступа (**access permissions**) и атрибут защиты объектов в памяти.

kaspersky

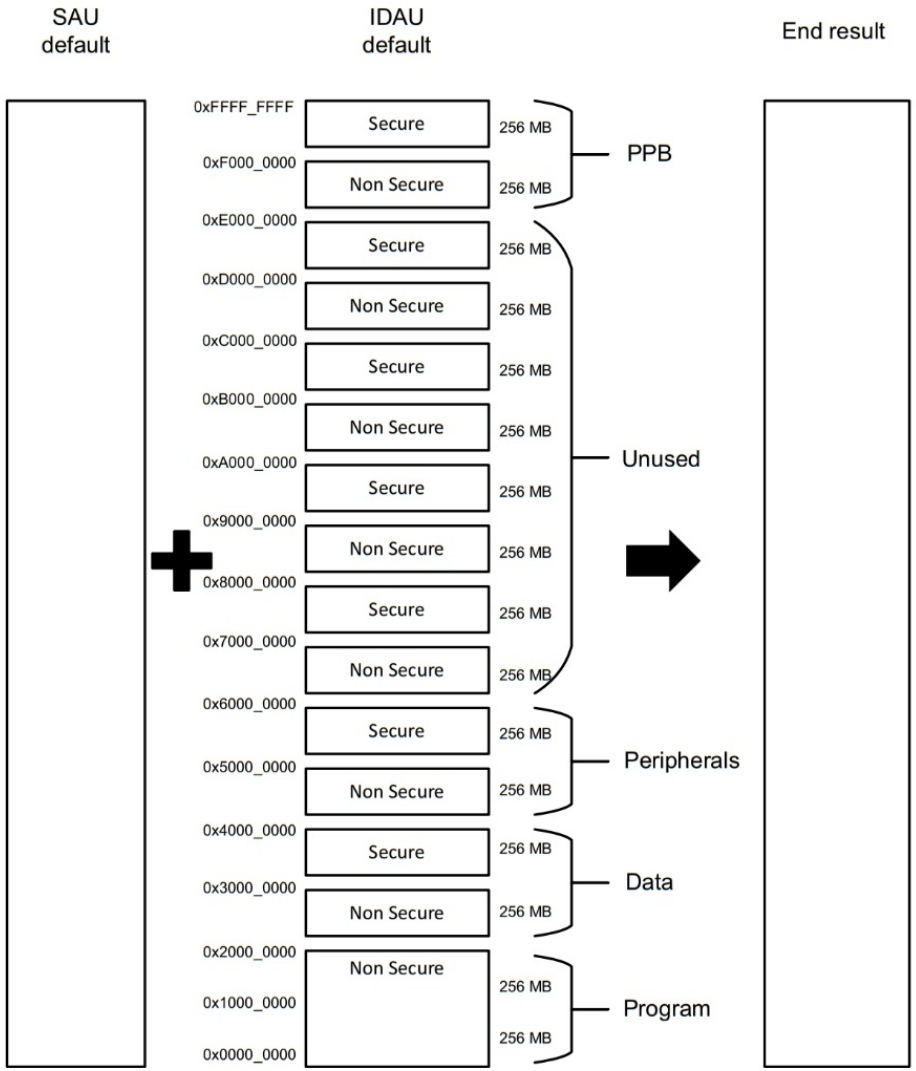
Подсистема защиты памяти. Блоки атрибутов защиты

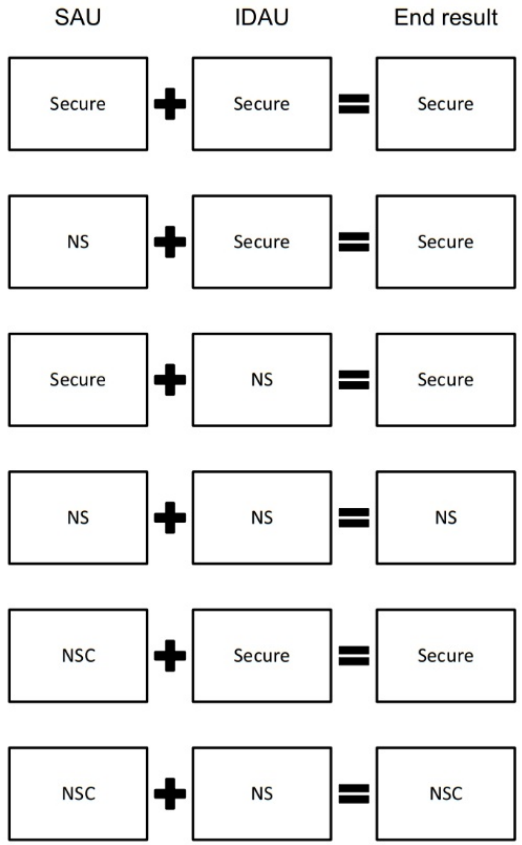


Пример распределения памяти с помощью блока IDAU

kaspersky

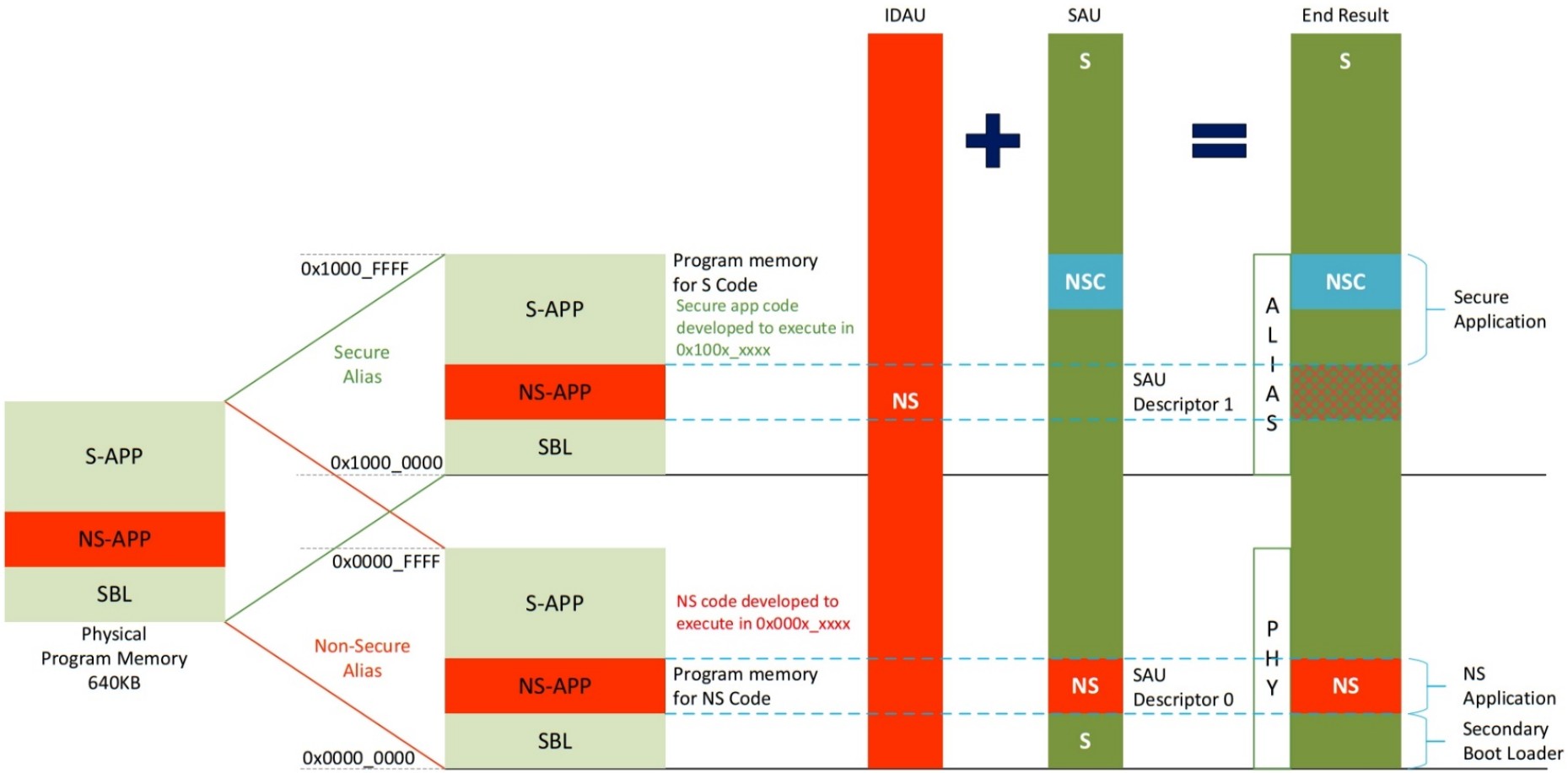
Подсистема защиты памяти. Блоки атрибутов защиты





Результат совместного использования блоков SAU и IDAU

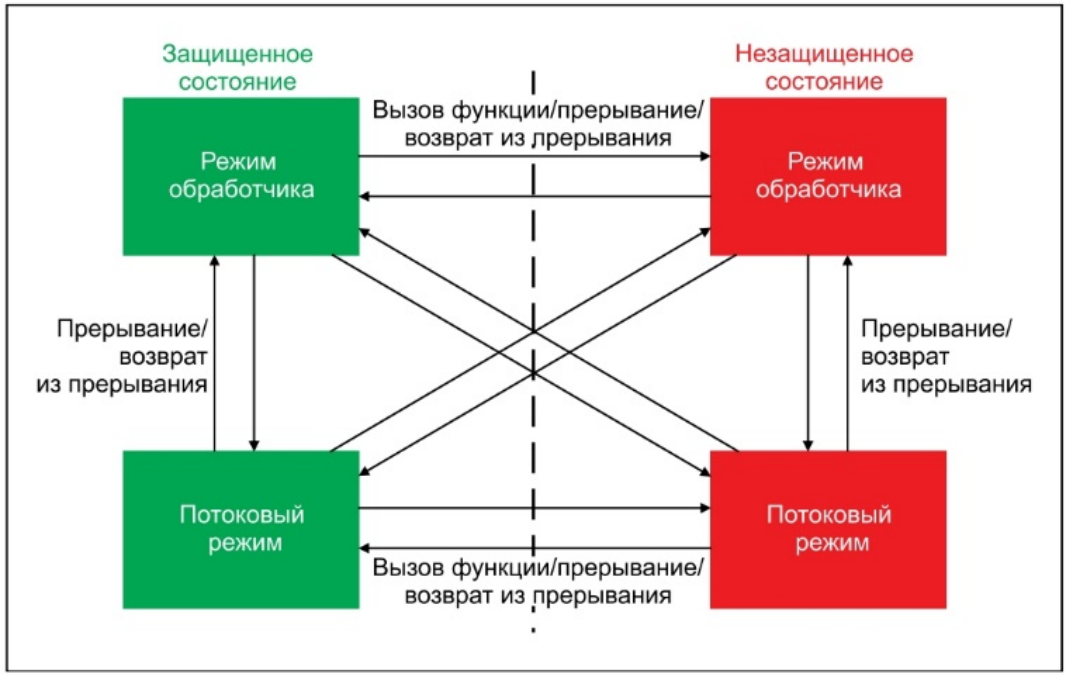
Подсистема защиты памяти. Блоки атрибутов защиты



Пример распределения памяти микроконтроллера путем совместного использования блоков SAU и IDAU

kaspersky

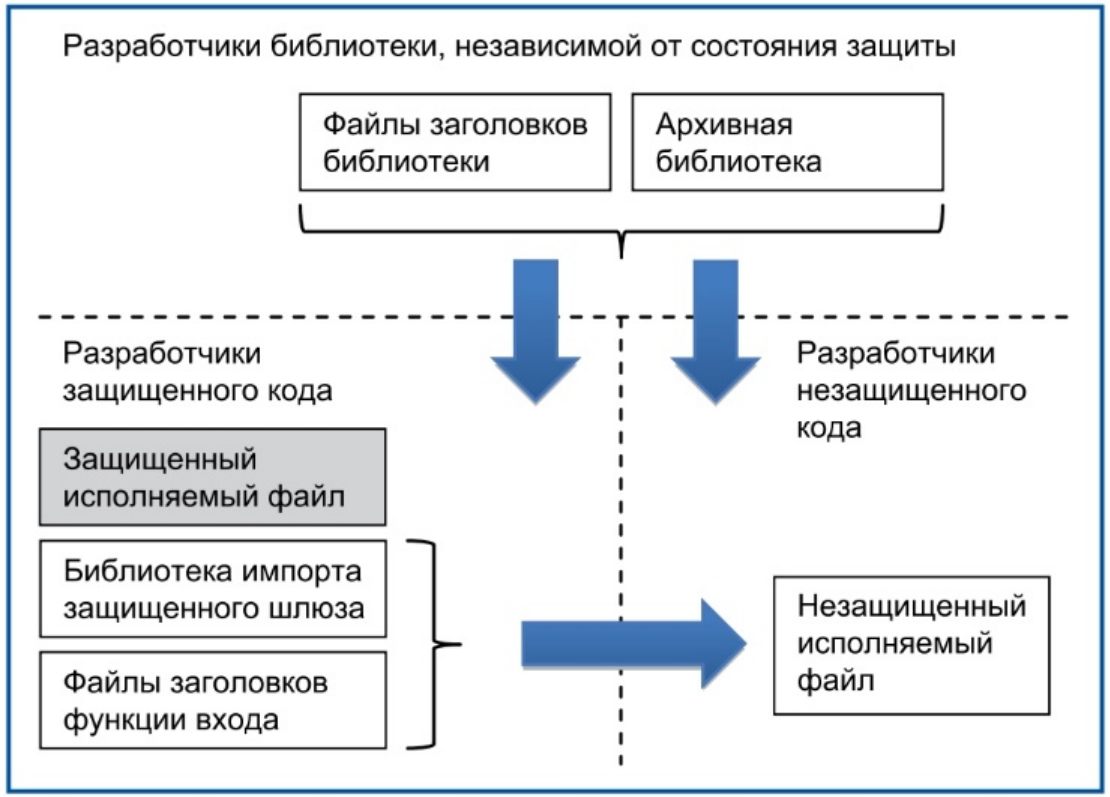
Прерывания



Переходы между состояниями осуществляются и при обработке исключений (**Exceptions**), и прерываний. Каждое прерывание конфигурируется как защищенное или незащищенное, только из защищенной области. Нет никаких ограничений относительного того, незащищенные или защищенные прерывания процессор будет обрабатывать при исполнении незащищенного или защищенного кода. Если возникшее исключение или прерывание имеет то же самое состояние защиты, что и текущее состояние защиты процессора, то последовательность обработки исключения (**Exception sequence**) всегда эквивалентна таковой для процессоров **Cortex-M** и обеспечивает низкое время реакции на прерывание. Основное отличие появляется, когда незащищенное прерывание возникает и обрабатывается процессором при исполнении защищенного кода. В таком случае для предотвращения утечки информации процессор автоматически помещает всю защищенную информацию (**Secure information**) в защищенный стек и стирает содержимое регистровых банков.

Технология **TrustZone** для **ARMv8-M** обеспечивает малое время реакции на прерывание, характерное для существующих процессоров **Cortex-M**. При этом время реакции на обработку незащищенных прерываний, возникающих в защищенном состоянии, становится несколько большим за счет необходимости размещения всего защищенного контента (**Secure contents**) в защищенном стеке.

Совместная разработка приложений для TZ Cortex-M



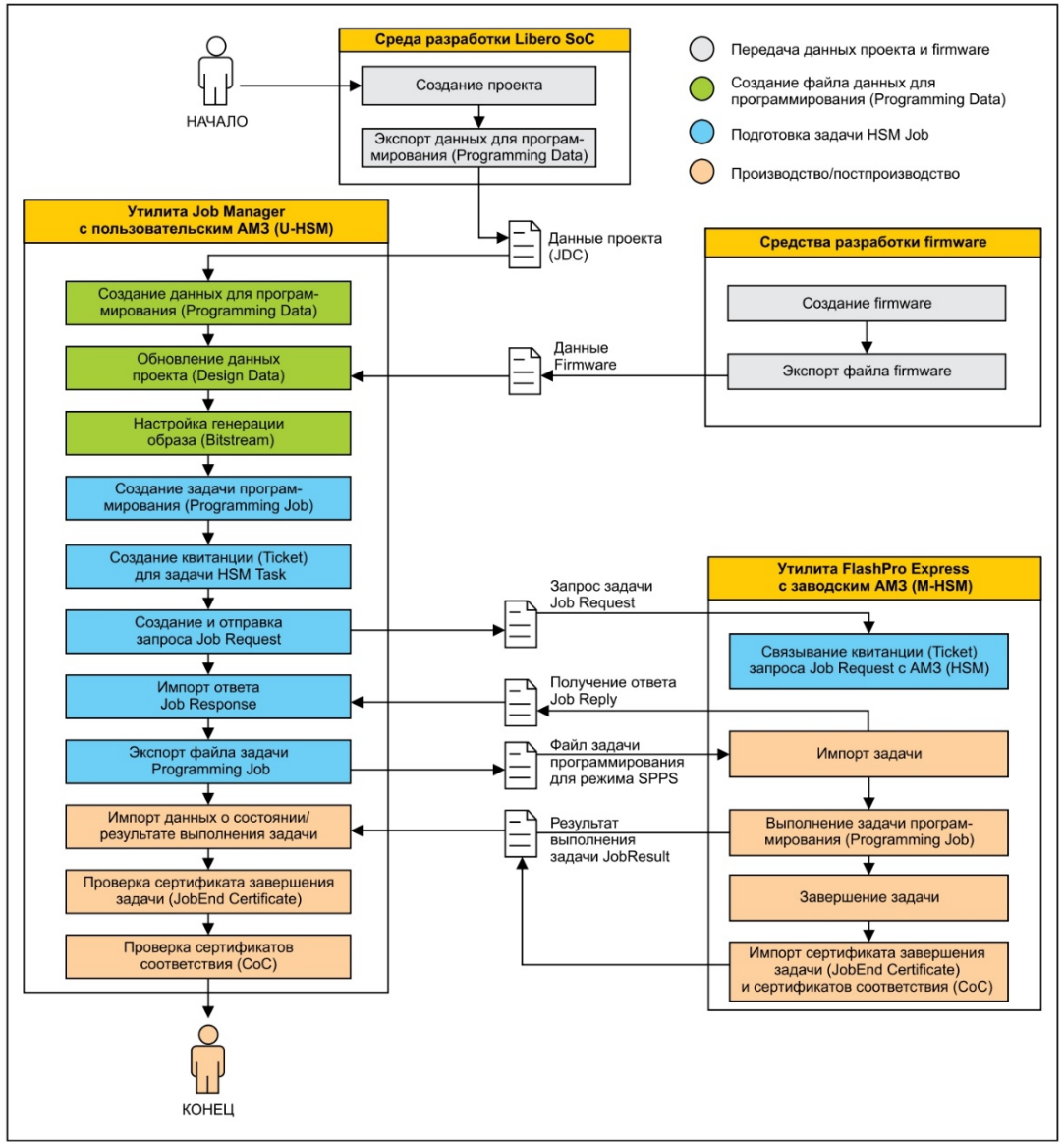
Существует два различных типа исполняемых файлов, по одному для каждого состояния защиты:

- В защищенном состоянии выполняется защищенный код из защищенного исполняемого файла (**secure executable**).
- В незащищенном состоянии выполняется незащищенный код из незащищенного исполняемого файла (**non-secure executable**).

Защищенные и незащищенные исполняемые файлы разрабатываются независимо один от другого.
С точки зрения незащищенного состояния вызов защищенного шлюза является вызовом регулярной функции (**regular function call**), как и возврат из вызова незащищенной функции. Поэтому необходимо, чтобы незащищенные исполняемые файлы могли разрабатываться инструментарием, которому ничего не известно о расширениях безопасности **CMSE3**.

Для разработки защищенных исполняемых файлов требуется, чтобы инструментарий поддерживал каждый вызов из незащищенного состояния, вызов или возврат в незащищенное состояние, а также обращения по адресу, предоставленному из незащищенного состояния. Двоичный интерфейс защищенного кода (**secure code ABI4**) всегда идентичен двоичному интерфейсу незащищенного кода (**non-secure code ABI**).

Защита производственного программирования с использованием АМЗ



Основная цель технологии – доверенное программирование чистых микросхем, защита от перепроизводства и обновления прошивки. Вся определенная пользователем информация о проекте, относящаяся к массиву прошивки, энергонезависимой памяти и защите, экспортируется из среды разработки посредством файла JDC и передается инженеру по эксплуатации, который использует, создает и передает в производство задачи программирования посредством АМЗ. На этом этапе можно обновить клиентские данные для энергонезависимой памяти, чаще всего предоставляемые командой разработчиков.

Все операции, относящиеся к генерации и применению ключей шифрования, передаче ключей и других защищенных данных, выполняются с помощью пользовательского АМЗ (U-HSM). На производстве выполняются предоставленные U-HSM задачи программирования. Производственный АМЗ (M-HSM) по защищенному протоколу обслуживает запросы от программатора и реализует политику защиты от перепроизводства.

Доказательства результатов программирования, такие как сертификат соответствия (CofC), сгенерированные запрограммированными микросхемами, и сертификаты завершения задач отправляются обратно инженеру по эксплуатации и могут быть проверены при помощи U-HSM.

С точки зрения безопасности, области разработки и функционирования всегда рассматриваются в качестве доверенного окружения. Инженер по эксплуатации может модифицировать политики безопасности, полученные из среды разработки, в JDC-файле. Основанное на M-HSM производство рассматривается как недоверенная область по отношению к проекту, политикам безопасности и выполнению задач.

kaspersky

Спасибо!

Андрей Самоделов
Системный аналитик
«Лаборатория Касперского»

Andrey.Samodelov@kaspersky.com

kaspersky