Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «**Информатика и системы управления**»

КАФЕДРА «**Программное обеспечение ЭВМ и информационные технологии**»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

НА ТЕМУ:

Метод программной реализации доверенной среды исполнения с помощью виртуализации процессоров архитектуры ARM

Студент группы **ИУ7-42М**		**А. В. Романов**
	(Подпись, дата)	(И.О. Фамилия)
Руководитель ВКР		**Д. Е. Бекасов**
	(Подпись, дата)	(И.О. Фамилия)
Нормоконтроллер		
	(Подпись, дата)	(И.О. Фамилия)

***2024** г.*

СОДЕРЖАНИЕ

РЕФЕРАТ

Расчетно-пояснительная записка к выпускной квалификационной работе «Метод программной реализации доверенной среды исполнения с помощью виртуализации процессоров архитектуры ARM» содержит 70 страниц, 4 раздела, 25 рисунков, 7 таблиц и список используемых источников из 37 наименований.

Ключевые слова: доверенная среда исполнения, виртуализация, безопасность, ARM, TrustZone, операционные системы, системное программирование.

Объект разработки: метод программной реализации доверенной среды исполнения

Цель работы: разработка метода программной реализации доверенной среды исполнения.

В аналитическом разделе представлен обзор реализаций доверенных сред исполнения и описаны их особенности. Сформулированы критерии оценки и проведено сравнение на их основе. Представлен обзор средств виртуализации в процессорных системах ARM. Представлена формализованная постановка задачи на разработку метода программной реализации доверенной среды исполнения с помощью виртуализации процессоров архитектуры ARM.

В конструкторском разделе разработан метод программной реализации доверенной среды исполнения с помощью виртуализации процессоров архитектуры ARM и представлено его формальное описание в виде диаграмм IDEF0 и схем алгоритмов. Выполнено проектирование ПО для реализации данного метода.

В технологическом разделе обоснован выбор средства программной реализации доверенной среды исполнения с помощью виртуализации процессоров архитектуры ARM. Разработано программное обеспечение, реализующее метод доверенной среды исполнения с помощью виртуализации процессоров архитектуры ARM.

В исследовательском разделе проведено исследование эффективности и применимости разработанного программного обеспечения. Выполнено сравнение результатов работы разработанного метода и метода с аппаратной поддержкой доверенной среды исполнения на базе процессоров с архитектурой ARM (ARM TrustZone).

Разработанный метод может найти применение в области серверных и облачных технологий.

ВВЕДЕНИЕ

Необходимость повышения безопасности исполнения приложений, работающих в системах безопасности и обрабатывающих защищаемую информацию, привела к разработке программно-аппаратных решений, создающих доверенные среды исполнения (англ. TEE – Trusted Execution Environment [1]) на базе аппаратных средств, доверенных загрузок или аппаратно-программных модулей доверенной загрузки. Intel [2] и ARM [3] являются лидерами в этой области. Целью данной работы является изучение существующих реализаций доверенных сред исполнения и разработка метода программной реализации доверенной среды исполнения с помощью виртуализации процессоров архитектуры ARM.

Для достижения поставленной цели необходимо решить следующие задачи:

- провести обзор существующих реализаций ДСИ;
- описать их достоинства и недостатки;
- сформулировать критерии сравнения и сравнить реализации;
- изложить особенности метода;
- представить формализацию в виде диаграмм IDEF0 и схем алгоритмов;
- выполнить проектирование ПО для реализации метода;
- обосновать выбор средств программной реализации;
- разработать ПО;
- провести исследование эффективности и применимости ПО.

1 Аналитический раздел

В данном разделе представлен обзор реализаций доверенных сред исполнения и описаны их особенности. Сформулированы критерии оценки и проведено сравнение на их основе. Представлен обзор средств виртуализации в процессорных системах ARM. Представлена оформализованная постановка задачи на разработку метода программной реализации доверенной среды исполнения с помощью виртуализации процессоров архитектуры ARM.

1.1 Анализ предметной области

В этом разделе представлен анализ предметной области. Описаны методы обеспечения защиты информации на современных процессорах. Даны понятие и характеристика доверенной среды исполнения.

1.1.1 Кольца привилегий

В целях безопасности, компоненты любой системы разделены на уровни привилегий – кольца защиты, за реализацию которых отвечает разработчик процессора. Во всех современных системах, реализована кольцевая система уровней привилегий. От внешнего кольца к внутреннему идёт увеличение полномочий для инструкций кода, выполняемых на процессоре в данный момент (рис. 1).

Можно создавать ещё более сложную систему – формировать ещё больше колец защиты, для ограничения каждого из компонентов системы. Однако, чем сложнее архитектура системы и чем больше количества кода в ней, тем проще злоумышленнику найти уязвимость и эксплуатировать её [4].

Главной задачей злоумышленника является получение доступа к привилегиям, которые бы позволили получить доступ к необходимым ресурсам системы. Может показаться, что архитектурно верным является решение размещать конфиденциальные данные исключительно на последнем кольце защиты – ведь получить доступ туда сложнее всего. Но у такого подхода есть свои недостатки [4]. Данный подход был переосмыслен – в настоящее используется

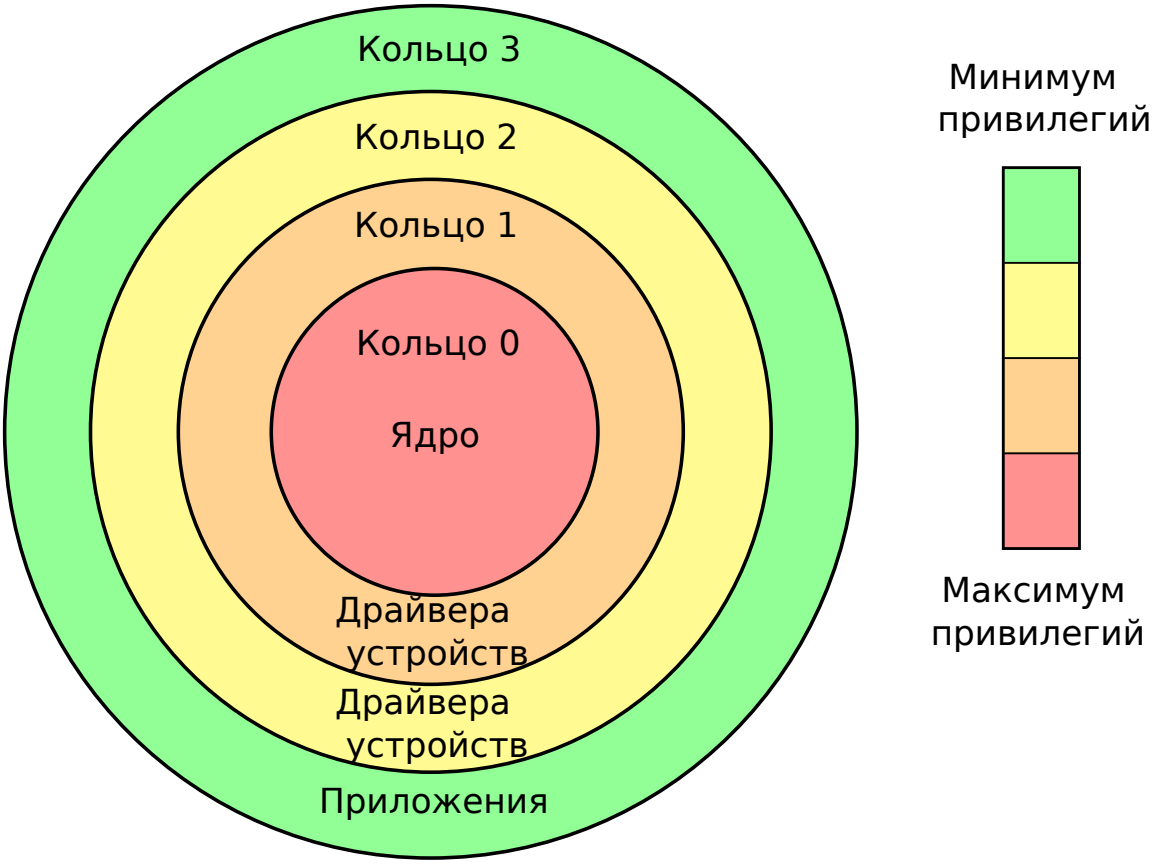

Рисунок 1 – Концептуальная схема представления колец защиты в современных системах

схема, когда и код, и конфиденциальные данные хранятся на одном и том же уровне, что и пользовательские предложения, однако, доступ к ним имеет только лишь процессор. Такой метод защиты информации называется анклавом или доверенной средой исполнения.

1.1.2 Доверенная среда исполнения

Доверенная среда исполнения (ДСИ) – специальная изолированная область, которая позволяет вынести из системы часть функциональности приложений и ОС в отдельное окружение. Содержимое памяти и выполняемый код в этой среде будут недоступны из основной системы, независимо от уровня текущих привилегий. Так, например, в ДСИ выполняется код отвечающий за реализацию различных алгоритмов шифрования, обработки закрытых ключей, паролей, процедур аутентификации и работы с конфиденциальными данными.

В случае, если система была скомпрометирована, информация хранящаяся в ДСИ не может быть определена, и доступ к ней будет ограничен лишь внешним программным интерфейсом. В отличии от других методов защиты защиты информации, таких как, например, гомоморфное шифрование, аппаратная реализация ДСИ практически не влияет на производительность системы и уменьшает время разработки программного обеспечения [1].

С другой стороны, аппаратная реализация ДСИ имеет свои недостатки:

— Некоторые современные процессоры имеют лишь частичную поддержку ДСИ, либо не имеют её вовсе.
— Отсутствует возможности программно исправить уязвимость. Найденные уязвимости в реализации ДСИ могут быть исправлены лишь в новых ревизиях процессора, т.е. без его физической замены, злоумышленник сможет эксплуатировать уязвимость.
— Увеличивается издержки производства на разработку таких процессоров – их конечная стоимость возрастает.

Таким образом, в некоторых случаях, появляется необходимость в программной реализации ДСИ. Стоит отметить, что в конечном счёте, программная реализация всё равно использует другие аппаратные механизмы предоставляемые процессором [5].

1.2 Существующие реализации ДСИ

1.2.1 ARM TrustZone

ARM TrustZone – технология аппаратного обеспечения ДСИ, разрабатываемая компанией ARM. Большинство процессоров разработанных ARM имеют поддержку TrustZone [5]. Данная технология основана на разделении режимов работы процессора на два ”мира”: обычный мир (Normal World) и безопасный мир (Secure World). Процессор переключается в безопасный мир по запросу (с помощью специальной инструкции), при работе с конфиденциальными данными. Всё остальное время, процессор работает в режиме обычного мира. Процессоры с поддержкой данной технологии имеют способность разде-

лять память, независимо от её типа, на ту, которая доступна только в безопасном мире, и ту, которую можно использовать в обычном мире. ARM предоставляют открытый исход программного обеспечения для поддержки данной аппаратной технологии – ARM Trusted Firmware [6].

Обычный и безопасный мир. Ключевой особенной ARM TrustZone является способность процессора переключаться между обычным и безопасным миром. Каждый из этих миров управляется собственной операционной системой, которые обеспечивают необходимую функциональность. Основное различие между этими ОС заключается в предоставляемых гарантиях безопасности. В один момент времени, процессор может находиться только в одном из двух миров, что определяется значением специального бита NS (Non-Secure), бит является частью регистра Secure Configuration Register (SCR). Этот регистр доступен для периферии только для чтения, изменять его значение может лишь сам процессор. Когда процессор находится в обычном режиме исполнения кода, значение бита NS равно 1, и наоборот, когда процессор находится в безопасном мире, значение бита NS равно 0.

За связь между обычным и безопасным миром отвечает специальный механизм – Secure Monitor. Он соединяет оба мира и является единственной точкой входа в безопасный мир. Для того, чтобы из обычного мира перейти в безопасный, существует специальная инструкция процессоров ARM – Secure Monitor Call (SMC). При вызове данной инструкции процессор передает управление Secure Monitor. Тот в свою очередь готовит систему к переходу из одного мира в другой и передает управление соответствующей ОС. Инструкция SMC используется как для перехода из нормального мира в безопасный, так и для перехода из безопасного в нормальный. Некоторые прерывания или исключения могут быть настроены так, чтобы они так же проходили через Secure Monitor и были обработаны в безопасном мире. ARM предоставляет спецификацию Secure Monitor Call Calling Convention (SMCCC) [7], которая является

стандартом при реализации вызовов SMC.

На рисунке 2 представлена концептуальная схема взаимодействия двух миров.

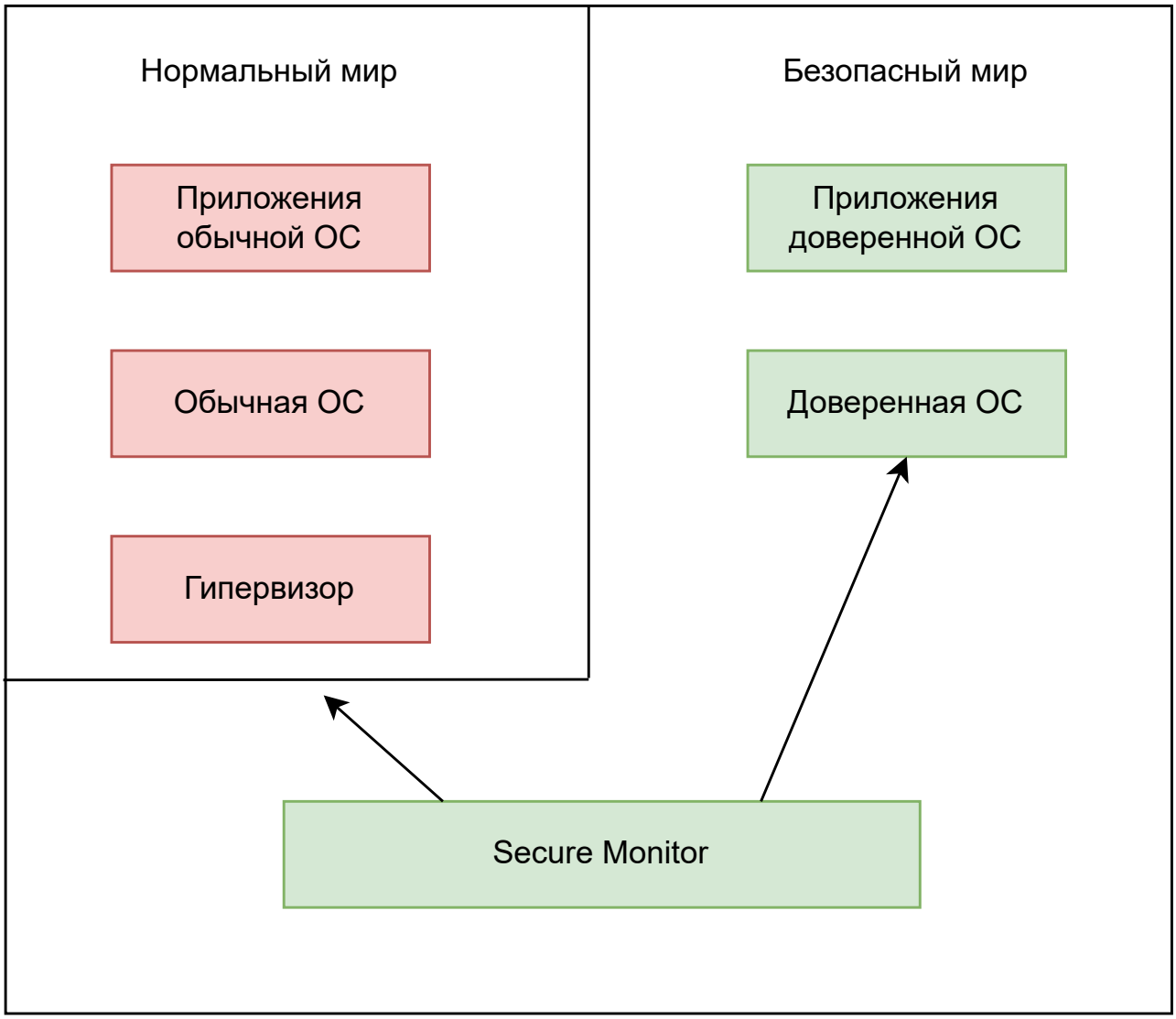

Рисунок 2 – Концептуальная схема взаимодействия двух миров для процессоров ARM Cotrex-A

Доверенная операционная система, так же как и обычная, может запускать приложения. Они, как и в обычном мире, обращаются к доверенной ОС при необходимости получения каких-либо ресурсов или обработки прерываний и исключений. Таким образом, Secure Monitor передаёт управление одному из доверенных приложений, и уже те, в свою очередь, обращаются к доверенной ОС.

ARM предоставляют открытый исходной код эталонной доверенной операционной системы, которая называется OP-TEE [8]. Global Platform предоставляет спецификацию для реализации API взаимодействия доверенных приложений [9] с доверенной ОС [10].

Физически миры разделены таким образом, что часть регистров доступны только в безопасном мире. Периферия, например память, может быть настроена так, что она может быть доступна лишь в определенном мире. Технология TrustZone в нормальном режиме работы процессора не позволяет программному обеспечению получить доступ к аппаратным средствам, которые могут быть доступны лишь только в безопасном мире.

При сборке компонентов системы, производитель устройства должен позаботиться о конфигурации периферии для работы с TrustZone:

— если предполагается, что периферия может получать доступ к безопасному режиму исполнения, процессор и внешнее устройство должны быть соединены (помимо различных шин) линией NS. Получение сигнал NS=0 от процессора к периферии означает, что команда является доверенной (например операция чтения или записи).
— В обратном случае, линия NS может быть опущена. Предполагается, что такая периферия не имеет никаких привилегий, т.е. NS=1 всегда.

На рисунке 3 представлена схема взаимодействия процессора и периферии для поддержки TrustZone. Периферийные устройства №1 и №3 соединены линией NS, а устройство №2 нет.

Стоит отметить, что чаще всего не вся периферия соединена сигналом NS с процессором. Например, тот факт, что производитель устройства не соединил сигналом NS камеру и ядра процессора, полностью исключает возможность предоставления пользователю доступа к устройству с помощью технологии распознавания лица.

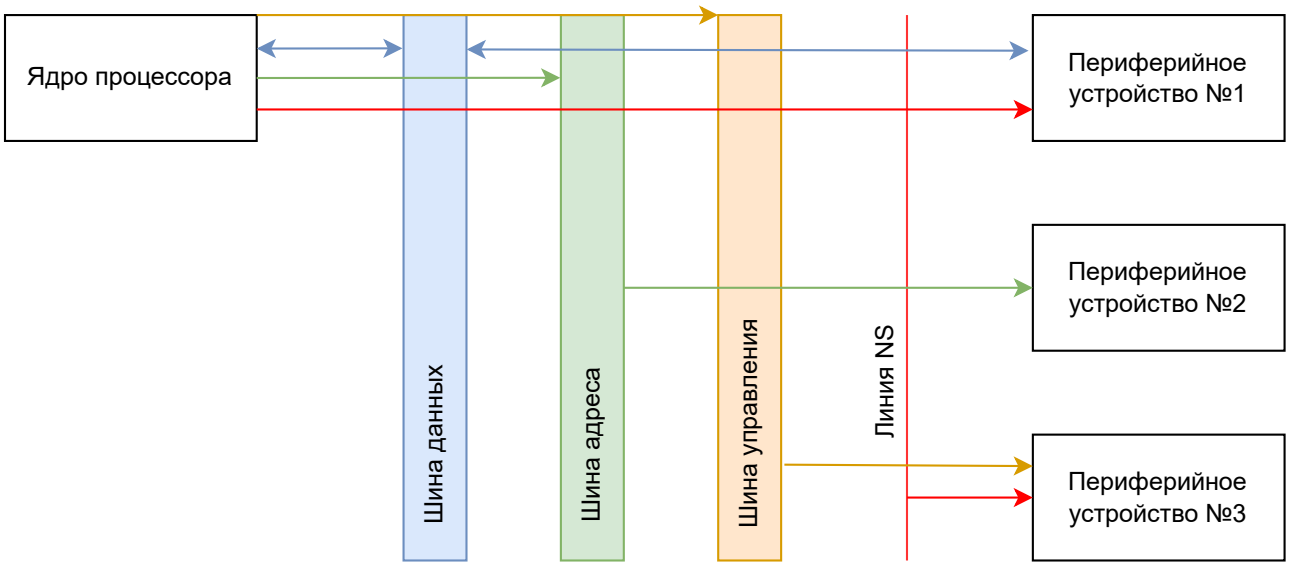

Рисунок 3 – Пример взаимодействия процессора и периферии для поддержки TrustZone

Режимы работы процессора. Современная архитектура ARM поддерживается три режима работы процессора:

- `EL0` – непривилегированный режим работы, предназначенный для исполнения обычных программ;
- `EL1` – привилегированный режим работы – исполняется кода ОС, обработчиков прерываний и исключений;
- `EL2` – режим работы гипервизора.

Для того, чтобы программа исполняемая на уровне EL0 могла перейти в EL1 (например, обратиться к ресурсам доступным только ОС), в архитектуре ARM существует команда Supervisor Call (SVC). Аналогично, Hypervisor Call (HVC) предназначена для перехода из режима EL1 в EL2.

Каждый из этих уровней могут исполняться в нормальных (Non-Secure) так и безопасных (Secure) режимах (рис. 4).

— Обычные приложения исполняются на уровне Non-Secure EL0, а приложения доверенной ОС на Secure EL0.

— Обычная ОС выполняется на уровне Non-Secure EL1. Все прерывания и исключения произошедшие в нормальном мире так же обрабатываются

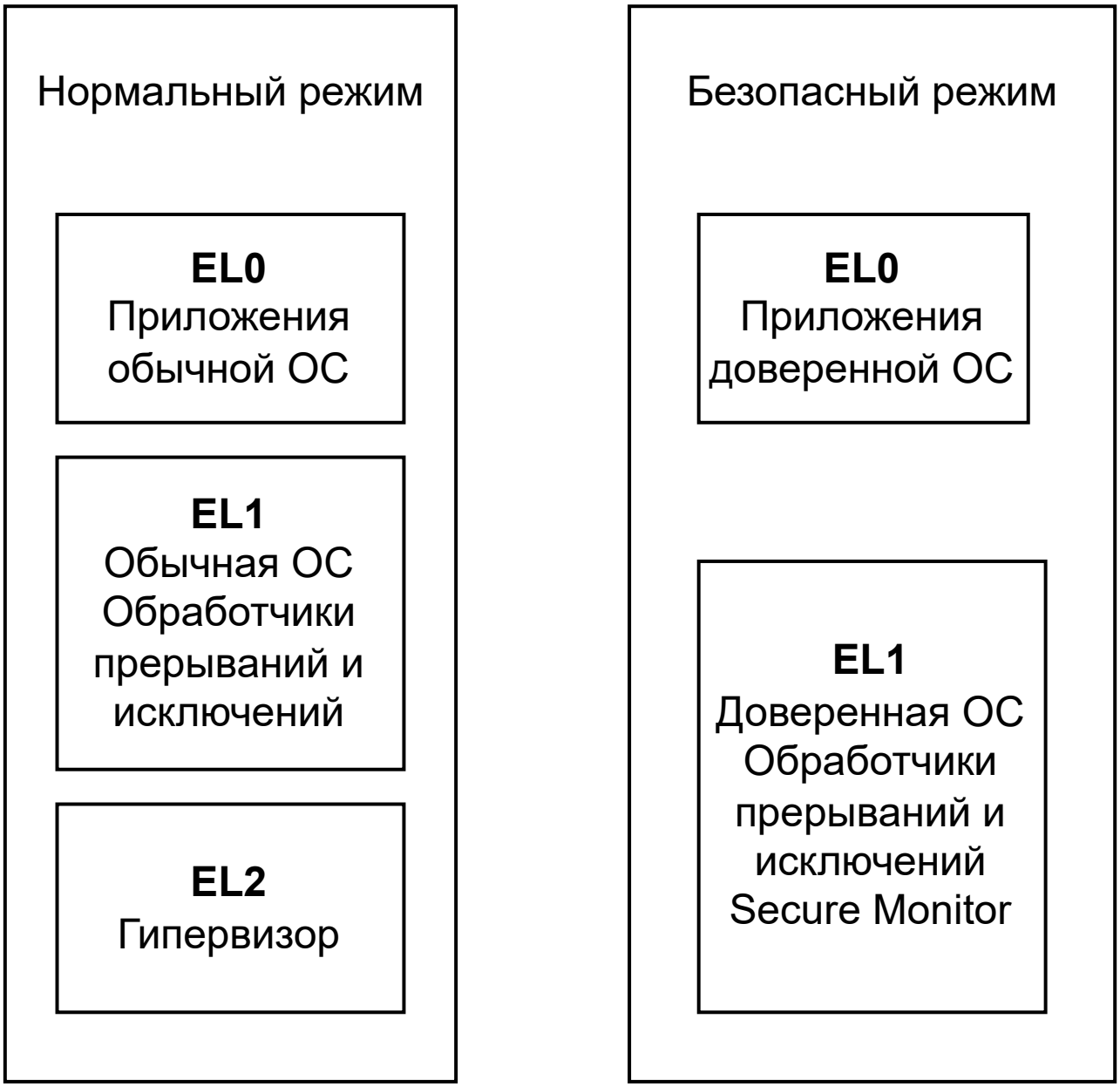

Рисунок 4 – Режимы работы процессоров архитектуры ARM

на этом уровне; доверенная ОС и прерывания произошедшие в безопасном мире исполняются на уровне Secure EL1.

— Secure Monitor всегда исполняется на уровне Secure EL1.

— Гипервизор исполняется в режиме Non-Secure EL2.

Бит NS, который был описан в предыдущей главе, определяет то, в каком режиме сейчас исполняется код: обычном (NS=1) или безопасном (NS=1).

Благодаря дублированию нормального и безопасного режима архитектура ARM позволяет запустить сразу две ОС: обычную и доверенную.

Разделение памяти. В целях безопасности в ARM TrustZone память разделяется на защищенную и незащищенную область. Благодаря контроллеру адресного пространства (TZASC – TrustZone Address Space Controller) незащищенная область память может использоваться только из обычного мира, а защищенная из безопасного. Кэш память так же разделяется на защищенную и незащищенную. Необходимо отметить, что данный контролер не является обязательным в архитектуры ARM TrustZone, поэтому разработчики могут принять решение отказаться от них в пользу более компактных и менее энергоемких устройств.

Проверка целостности. Проверка целостности ДСИ является важным механизмом, который не позволит злоумышленнику изменить исходный код доверенной ОС или доверенных приложений. В ARM TrustZone данный механизм реализован на аппаратном уровне как для ОС, так и для приложений: каждый раз, при загрузке доверенной ОС в память, с помощью цифровой подписи проверяются её целостность. Аналогичная схема используется и при загрузке доверенных приложений. Запустить ОС может только подписанные приложения, подпись формируется разработчиками, на стадии компиляции в исполняемый файл.

На данный момент, доверенная среда исполнения от компании ARM не поддерживает процедуру удалённой проверенной проверки (например, с помощью сервера). Существуют лишь программные решения этой проблемы от сторонних разработчиков [5].

1.2.2 Intel SGX

Intel Software Guard Extension (SGX) – реализация ДСИ от компании Intel, включена в большинство современных процессоров Intel Core. Конфиденциальность и целостность в этой технологии достигается с помощью использования анклава – специальной, зашифрованнаой области кода и данных. Это достигается с помощью различных компонентов и протоколов, одним из которых является специальная область памяти, называемая Processor Reserved Memory

(PRM), обеспечивающая безопасное хранилище, к которому не может обращаться никто, кроме самого процессора. Для выполнения кода, после череды его проверок, он загружается извне в PRM. Процессор с помощью специальных инструкций переходит в режим анклава (enclave mode) и выполняет загруженный код. В технологии Intel SGX именно анклав является доверенной средой исполнения.

Processor Reserved Memory. Processor Reserved Memory (PRM) – защищенная часть оперативной памяти, к которой не имеет доступ код, который исполняет в режиме non-enclave. Это реализуется аппаратно, с помощью специальных контроллеров доступа к памяти.

Данный участок памяти подразделяется на дополнительные разделы:

— Encalve Page Cache (EPC) – разделенная страницы размером 4Кб область памяти, которые хранят код конкретного анклава, к которому относятся; это позволяет использовать в системе несколько анклавов одновременно. EPC управляется программным путём с помощью ОС. Доступ к нему может быть получен только программным обеспечением входящим в данный анклав.

— Enclave Page Cache Map (EPCM) – массив с одной записью для каждой страницы, хранимой в EPC; запись содержит в себе метаданные этой страницы: владелец, виртуальный адрес и т. д.

— SGX Enclave Control Structure (SECS) – содержит в себе метаданные соответствующего анклава; хранится в специальной страничной части EPC. Страница, ассоциированная с SECS не отображается в память напрямую и доступна только для реализации SGX – это сделано в целях дополнительной безопасности.

Схема представления памяти с использованием Intel SGX представлена на рисунке 5.

Анклав получает доступ к EPC, выделяя часть своей виртуальной памяти

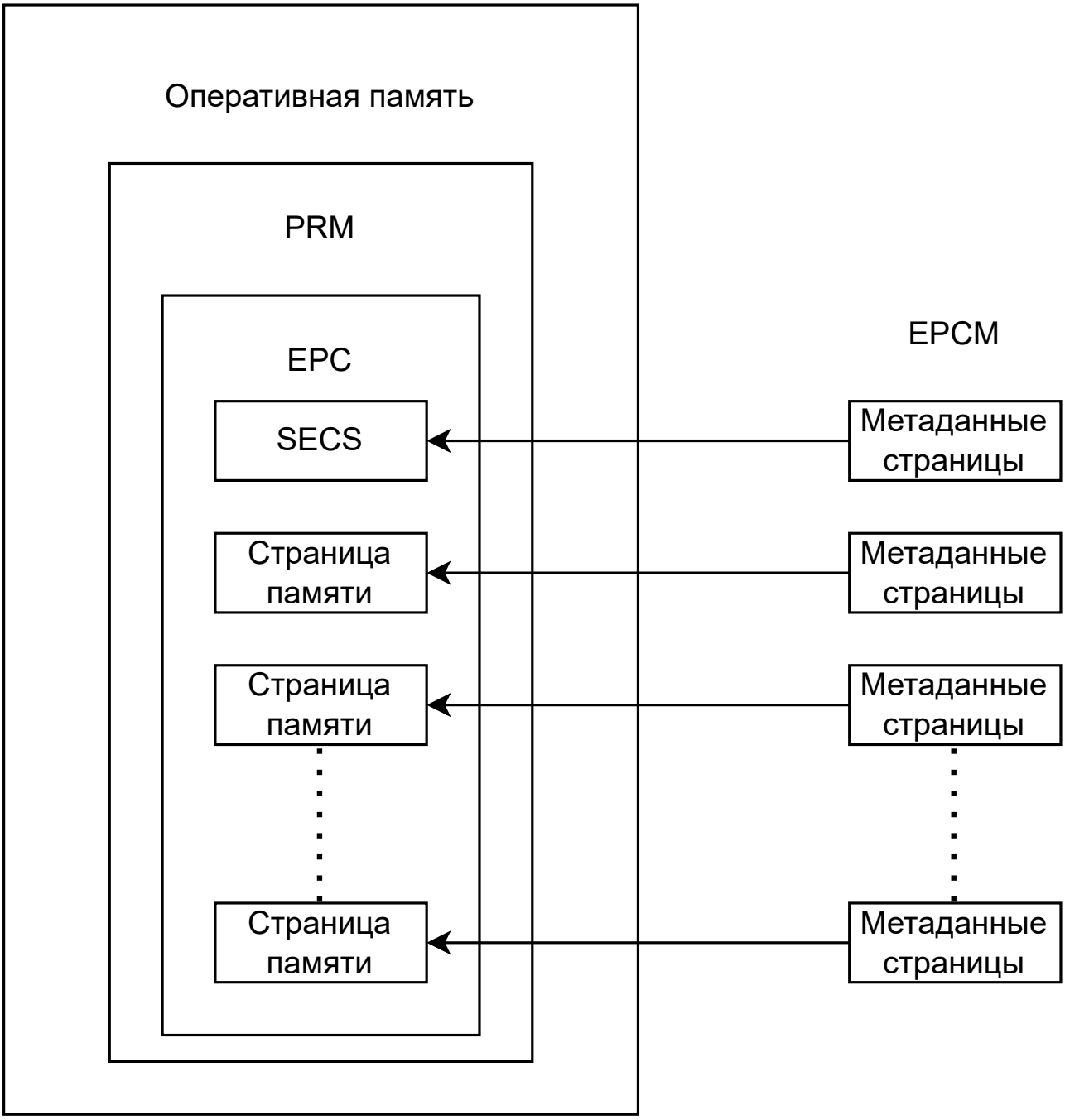

Рисунок 5 – Схема представления памяти Intel SGX

под линейный диапазон адресов анклава (Enclave Linear Address Range, ELRANGE), который содержит адреса сопоставленные с EPC. Другие адреса виртуальной памяти отображаются на память расположенную за пределами EPC. Процессор проверяет что в результате трансляции физического адреса страницы, находящейся в EPC, виртуальный адрес совпадает с адресом хранящимся в EPCM – это позволяет предотвратить атаки на трансляцию адресов.

Каждая страница обладают индивидуальными правами доступа, которые устанавливаются при выделении соответствующей страницы и определяются автором анклава, что так же является дополнительной мерой безопасности. Страницы разделены на те, которым разрешено чтение, запись и выполнение кода анклава. Эта информация так же находится в метаданных страницы, хранящихся EPCM.

State Save Area (SSA) – ещё один компонент Intel SGX, отвечающий за сохранение текущего состава анклава. Это специальная область памяти, которая используется для хранения контекста выполнения кода анклава. Эта область памяти необходима, например, когда в системе произошло прерывание – процессору необходимо перейти в нормальный режим исполнения для его обработки. После этого, процессор может загрузить состояние анклава из SSA и возобновить выполнение кода.

Ещё одной особенностью с точки зрения безопасности и производительности, является возможность вытеснения страниц из PRM в обычную (не-PRM) память. Большинство современных компьютеров поддерживают избыточное использование оперативной памяти, выгружая некоторые страницы во вторичные устройства. Технология Intel SGX поддерживает это, добавляя меры, которые гарантирую целостность и конфиденциальность вытесняемых страниц: вытесняемая страницы шифруется с помощью симметричного шифрования, а ключ хранится в специально отведенных для этого страницах EPC.

Жизненный цикл анклава. Для управления анклавами Intel предоставляет набор специальных инструкций:

— `ECREATE` – создаёт новый анклав и сохраняет его метаданные в SECS.

— `EADD` – используется для добавления новых страниц в анклав; ОС загружает новый страницы в EPC, заполняя необходимые метаданные. Вызывать эту инструкцию можно только после создания анклава, попытка добавить страницы после этого этапа приведёт к ошибке.

— С помощью инструкции EINIT и специального токена от Launch Enclave, процессор начинает выполнять код анклава. Launch Encalve – специальный анклав, проходящий все те же стадии инициализации, что и другие. Его основная цель является выдача токена другим анклава на основе списка одобренных анклавов.

— Приложения могут выполнить команду EENTER для входа в режиме анклава и выполнения там своего кода, по окончанию выйти оттуда используя команду EEXIT. В виртуальном адресном пространстве этих приложений должны иметься соответствующие страницы EPC.

— Команда AEX используется при возникновении прерывания или исключения, после её выполнения процессор переходит в обычный режим исполнения, сохраняя контекст в SSA.

— ERESUME – возобновить выполнение анклава с контекстом, сохраненным в SSA. Стоит отметить, что анклав может иметь более одного SSA в случаях, если при выполнении одного и того же блока происходит несколько прерываний.

После выполнения кода, в метаданных страниц, ассоциированных с этим анклавом, выставляется пометка, что они невалидны. После этого очищается TLB кэш. Это позволяет защитить Intel SGX от атак на память. На рисунке 6 представлена схема жизненного цикла анклава с использованием команд, описанных выше.

Проверка целостности. В Intel SGX реализована поддержка механизма локальной и удаленной проверки целостности.

Локальная проверка используется для установления канала связи, который гарантирует конфиденциальность, двумя анклавами на одном устройстве. Для обмена симметричными ключом используется протокол Диффи-Хеллмана [11]. Локальная проверка начинается с того, что один из анклавов отправляет значение MRENCLAVE (индивидуальный идентификатор анклава) друго-

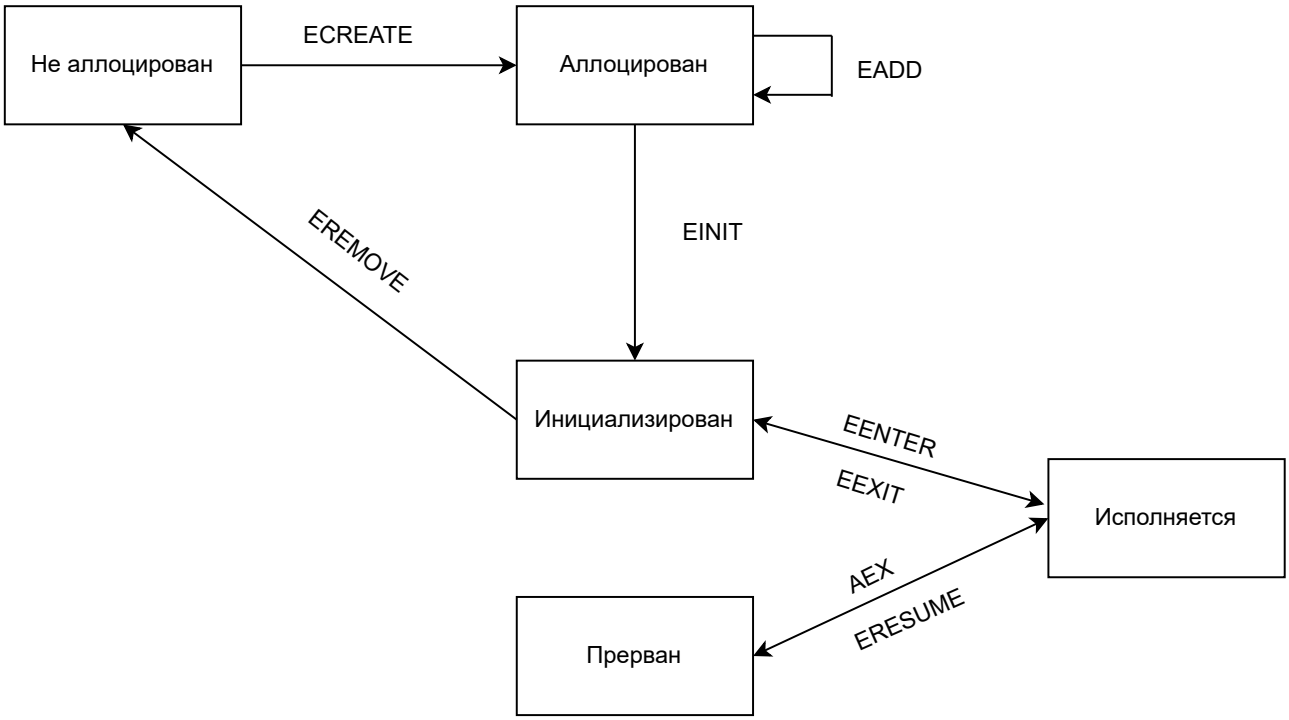

Рисунок 6 – Жизненный цикл анклава

му анклаву, который находится на том же устройстве. Отправитель называется верификатором, а получающий анклав утверждающим. Отправитель использует полученное от верификатора значение MRENCLAVE для создания отчёта (claimer), который он отправляет обратно верификатору. Отчёт может быть проверен с помощью специально ключа REPORT KEY, который хранится на устройстве и доступен всем анклавам. Он так же содержит данные обмена ключами Диффи-Хеллмана, который в дальнейшем будут использованы для создания защищенного канала связи. После проверки отчёта, верификатор создает и отправляет отчёт для утверждающего анклава. Затем обе стороны могут создать защинный канал используя данные Диффи-Хеллмана, содержащиеся в обоих отчетах. На рисунке 7 представлена схема локальной проверки целостности в Intel SGX.

Удаленная проверка целостности реализовывается с помощью удаленной службы Intel Attestation Service [13]. В процессе проверки, используется подсчёт хэш-суммы анклава с помощью хэш-фукнции SHA-2 [12], специального анклава находящегося в однократно записываемой памяти и ключей, которые

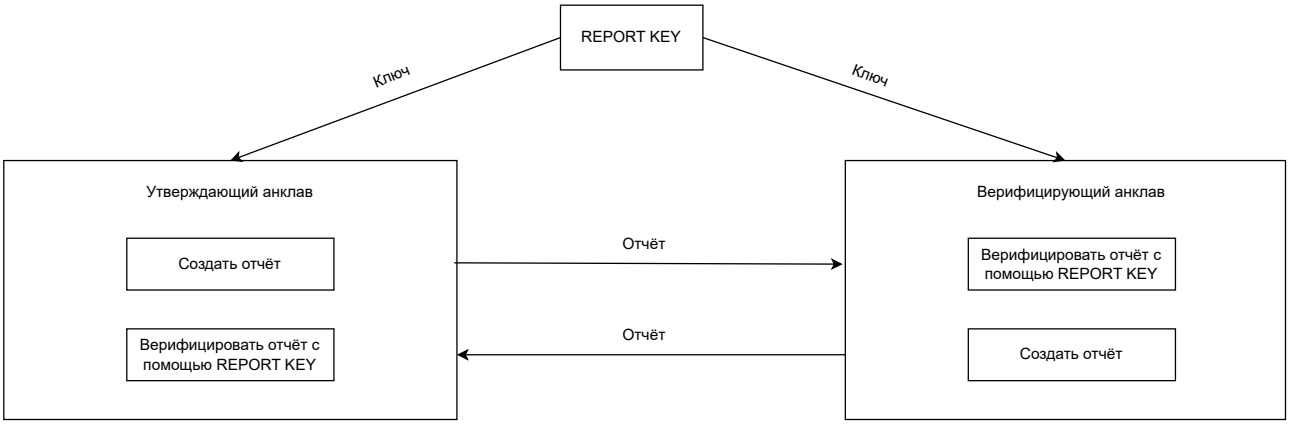

Рисунок 7 – Схема локальной проверки целостности в Intel SGX

были расположены в устройстве на стадии его производства. С помощью ключей и хэш-суммы, анклав расположенный в однократно записываемой памяти формирует специальный отчёт, отправляемый в службу Intel Attestation Service, и та, в свою очередь, на основе этого отчёта проверяет целостность анклава.

1.2.3 Keystone

Keystone – реализация доверенной среды с открытым исходным кодом для процессоров на базе архитектуры RISC-V. В отличии от Intel SGX и ARM TrustZone, эта технология является полностью программным решением с открытым исходным кодом, построенным на использовании аппаратных особенностей архитектуры RISC-V. Keystone предоставляет спецификацию для разработчиков устройств, выполнение которой гарантирует поддержку этого механизма [14].

Компоненты системы. Keystone состоит из нескольких компонентов, каждый из которых выполняются на разном уровне привилегий [15].

Всего есть три уровня привилегий:

- `U-mode` (user) – режим исполнения пользовательских процессов;
- `S-mode` (supervisor) – режим выполнения кода ядра;
- `M-mode` (machine) – режим, в котором осуществляется доступ к периферии устройства.

Ниже будут описаны компоненты, с помощью которых строится доверенная среда исполнения Keystone.

Trusted Hardware – совместимые со спецификацией Keystone ядра архитектуры RISC-V и ключи (открытый и закрытый), используемые для подписи анклава. Аппаратное обеспечение также может содержать дополнительные функции, например разделение кэша, шифрование памяти, криптографический защищенный источник случайных чисел.

Security Monitor (SM) – выполняется в режиме M. Предоставляет интерфейс для управления жизненным циклом анклава, а так же для использования специфических возможностей платформы. SM обеспечивает выполнение гарантий безопасности Keystone, поскольку он отвечает за изоляцию анклавов и обычной (недоверенной) ОС.

Анклавы представляет собой среду, изолированную от операционной системы и других анклавов. Каждому анклаву выделяется отдельная область физической памяти, получить доступ к которой может только он сам и Security Monitor. Каждый анклав состоит из анклавного приложения, выполняемого на уровне пользователя (режим U) и Runtime (режим S).

Приложения анклава (EAPP) – приложение пользовательского уровня, которого выполняется в анклаве. Можно создавать собственное приложение с нуля или просто запустить существующий исполняемый файл.

Runtime – программное обеспечение выполняющиеся в режиме S, реализующие такие следующий возможности: системные вызовы, управление виртуальной памятью, обработка прерываний и так далее.

На рисунке 8 представлена концептуальная схема компонентов системы ДСИ Keystone.

Жизненный цикл анклава. Анклав Keystone может находится в трех состояниях [15].

1) Создание. Анклав загружается на непрерывный диапазон физической па-

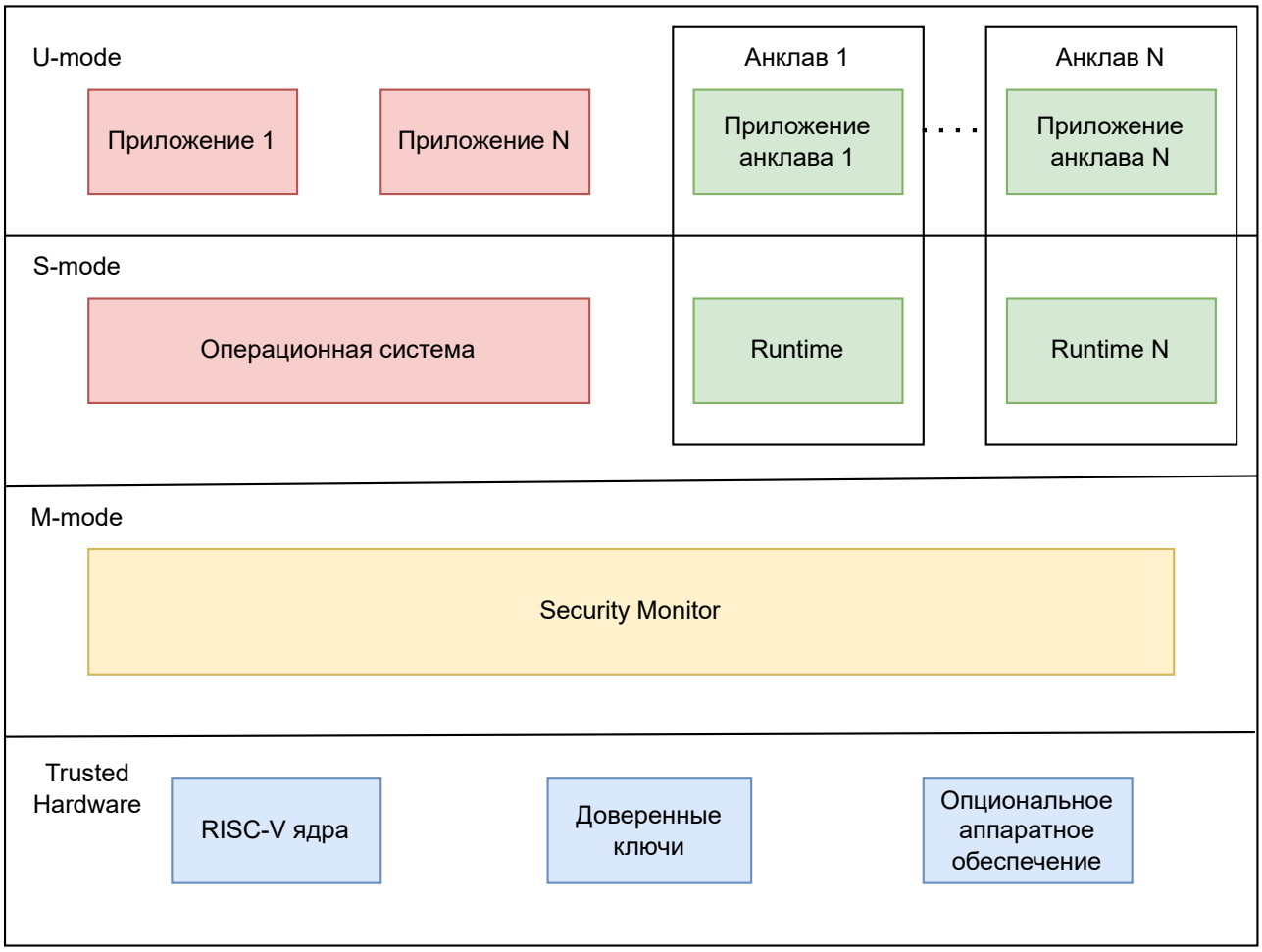

Рисунок 8 – Компоненты системы доверенной среды исполнения Keystone

мяти, которая называется приватной памятью анклава. Недоверенный код (например, операционная система) выделяет эту память и инициализирует таблицу страниц анклава, загружает код компонента Runtime и приложение анклава. Для создания анклава вызывается Secure Monitor, которые изолирует и защищает приватную память с помощью механизма защиты физической памяти, что позволяет защитить память анклава от изменений и чтений для любого ядра процессора. После выполнения этих действий, SM помечает анклав готовым для выполнения.

2) Выполнение. Недоверенный код запрашивает у SM разрешение на исполнение кода анклава на одном из ядер процессора; SM выдает разрешение на выполнение и ядро начинает выполнять его код. Процесс выполнения может быть прерван (например для обработки прерывания), и, в таком

случае, ядро перестанет выполнять код анклава и уберёт разрешение на выполнение его кода.

3) Разрушение. Недоверенный код может уничтожить анклав в любое время. При его уничтожении, SM освобождает выделенную память для анклава, снимает с неё защиту и передаёт её в распоряжение операционной системы, предварительно заполнив нулями её содержимое.

На рисунке 9 представлена схема жизненного цикла анклава Keystone.

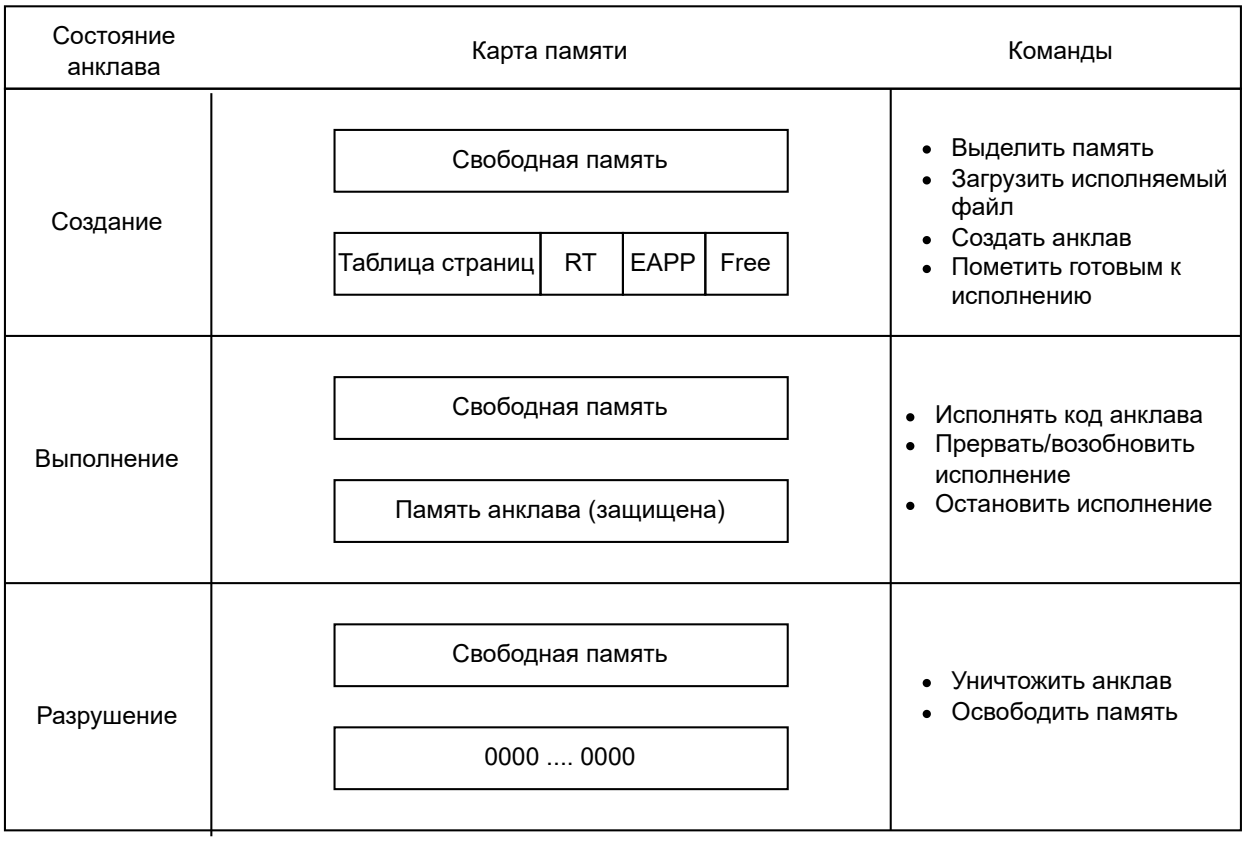

Рисунок 9 – Жизненный цикл анклава

При создании анклава, участок его памяти состоит из таблицы страницы, компонента Runtime, приложения и свободного участка памяти. При его уничтожении, она предварительно заполняется нулями.

Проверка целостности. Архитектура ДСИ Keystone предполагает использование как локальной, так и удаленной проверки целостности анклава.

Для того чтобы локально проверить целостность анклава, при его созда-

нии вычисляется хэш-сумма (на основе кода и данных). Далее, анклав генерирует ключ шифрования, который будет использоваться для аутентификации анклава и защиты его кода и данных. Ключ подписывается с помощью доверенного ключа (см. рис. 8), для удостоверения его подлинность. Ключ анклава и информация о нем, включая его хэш-сумму и сертификат, передаются клиенту (например, хост-системе), который будет проверять анклав. Клиент проверяет аутентичность хэш-суммы анклава и его ключа, используя доверенный открытый ключ. Если анклав и его ключ прошли проверку, клиент может быть уверен в его подлинности и целостности.

1.3 Виды угроз безопасности

Существует множество различных видов атак, которые могут быть направлены на компрометацию доверенных сред исполнения. В конечном итоге, они приводят к опасности для целостности кода и данных, хранящихся внутри ДСИ. В этом разделе будут описаны наиболее распространенные виды атак.

1.3.1 Физические атаки

Физические атаки обычно связаны с использованием аппаратных средств или их эксплуатацией [18]. Ниже будут приведены наиболее распространенные типы физически атак на устройство.

Самая простая атака типа отказ в обслуживании, заключается в отключении питания компьютера или другого электрического оборудования и предотвращении его использования. Более продвинутые методы включают в себя подключение USB накопителей и загрузка устройства таким образом, чтобы получить доступ к периферии и кражу ключей шифрования.

Прослушивание шины [18] – контролируя шину на материнской плате компьютера, злоумышленники могут подслушивать трафик, и возможно, даже его модифицировать, добавляя, удаляя или воспроизводя старые новые.

Другой подход к физическим атакам заключается в мониторинге энергопотребления процессора и вывод о типе вычислений, выполняемых в данный момент. Такой тип атаки называется атакой анализа энергопотребления (англ.

Power Analysis Attack [17]). Он может быть использован против ДСИ, поскольку легко выявить паттерн их вычислений.

Атаки на чип [18] – ещё один вид физической атаки. В этих атаках используются ионно-лучевые инструменты для атак на микросхемы. Суть этих атак заключается в изменении поведения аппаратных средств микросхем таким образом, чтобы обходить механизмы защиты. Например, с помощью такого вид атак, можно добиться чтобы процессор пропускал некоторые инструкции ПО.

1.3.2 Атаки на привилегированное ПО

Атаки на привилегированное ПО является одной из главных проблем всех ДСИ. Атаки этого типа предполагают, что они могут происходить со всех уровней привилегий. Такие типы атак включают в себя получение доступа к режиму управления всей системой, т.е. получение доступа к наиболее привилегированному режиму работы ПО в системе. Раньше, доступ к такому режиму был возможен только с помощью аппаратных средств, но с недавнего времени и с помощью программных [5], что позволяет атакам этого типа эксплуатировать это.

1.3.3 Программные атаки на периферию

Другой формой атаки является использование периферийных устройств или их интерфейсов для получения доступа к памяти или для реализации других атак. Такие атаки не требуют аппаратного обеспечения или физического воздействия на устройство.

Примером такой атаки является атака rowhammer [19]. Суть атаки заключается в многократной записи в одну и ту же ячейку памяти, что приводит к изменению значения соседних ячеек, т.е. эксплуатация аппаратной особенности. Многократно изменяя значения определенных адресов памяти, злоумышленник может изменить структуры данных, отвечающие за безопасность систем, таким образом, чтобы получить привилегии суперпользователя.

Другой разновидность подобных атак является злоупотребление определенными функциями ПО в злонамеренных целях. Например, использование

функций мониторинга производительности и температуры процессоров для получения информации об его активности и текущих вычислений.

1.3.4 Трансляция адресов

Другой серьезной проблемой для ДСИ являются атаки на трансляцию адресов, особенно для Intel SGX и Keystone, в которых процесс трансляции адресов управляется недоверенным системным ПО, в отличии от ARM TrustZone (управляется доверенной ОС).

Одним из типов данного вида атаки заставляет ОС перестроить память таким образом, чтобы выполнить нежелательные инструкции. Например, ОС может менять местами содержимое двух ячеек памяти, содержащих разные наборы инструкций, причём один набор из наборов является вредоносным. В результате, атакованное приложение выполнит вредоносные инструкции, что может привести к раскрытию ключей шифрования, пользовательских данных и другой конфиденциальной информации.

Ещё одни слабым местом является вытеснение памяти из защищенной в незащищенную. Когда память перераспределяется в защищенную, ОС может без каких-либо проверок загрузить её и передать управление приложению, которое начнёт исполнять инструкции. ДСИ должны быть защищены от таких атак, отслеживая корректный виртуальный адрес для каждой физической ячейки и связывать каждый вытесняемый фрагмент памяти к правильному виртуальному адресу. Однако, даже этого может оказаться недостаточным, потому что злоумышленник может воздействовать на TLB-кэш, хранящий результаты транслирования адресов [18].

1.4 Сравнение реализаций ДСИ

1.4.1 Критерии сравнения

Для сравнения раннее описанных реализаций ДСИ были выделены следующие критерии:

- К1 – безопасность;
- К2 – производительность (место);

- К3 – полнота и надежность механизмов проверки целостности ДСИ;
- К4 – является проприетарным решением;
- К5 – программное или аппаратное решение.

1.4.2 Сравнение безопасности

В таблице 1 приведено сравнение защиты ДСИ от видов атак, описанных в разделе 1.3.

Таблица 1 – Результаты сравнения безопасности ДСИ

	ARM TrustZone	**Intel SGX**	**Keystone**
Физические атаки	-	-	-
Атаки на привилегированное ПО	+	+	+
Программные атаки на периферию	+	+	+
Трансляция адресов	+	+	+

— Ни одна из реализаций ДСИ, рассмотренных в данной работе, никак не защищена от физических атак.

— Intel SGX и Keystone защищены от атак на привилегированное ПО, потому что оно не является доверенным и не может получить доступ к памяти анклава. Разделение на доверенную и недоверенную ОС в ARM TrustZone так же позволяет сделать вывод о том, что данная ДСИ защищена от данных видов атак.

— В случае Intel SGX и Keystone, проверка целостности не позволяет достичь программным атакам на периферию желаемых результатов. В ARM TrustZone защита достигается с помощью разделения ячеек памяти на доверенную и не доверенную.

— Во всех рассмотренных реализациях ДСИ имеются механизмы от предотвращения атак с использованием трансляции адресов: сбор TLB-кэша, шифрование вытесняемых страниц и т.д.

Можно сделать вывод, что рассмотренные реализации ДСИ одинаково защищены от атак, описанных в разделе 1.3, но, с помощью разных механизмов защиты.

1.4.3 Сравнение производительности

В этом разделе будет проведено сравнение производительности доверенных сред исполнения. Все операции, описанные ниже, производились непосредственно во время исполнения кода ДСИ: из доверенной ОС (ARM TrustZone) и из анклавов (Intel SGX, Keystone).

Характеристики устройств, на которых производилось тестирование производительности, представлена в таблице 2.

Таблица 2 – Характеристика устройств

	Процессор	**Ядер**	**Память (Гб)**	**ПО**
Raspberry Pi3 B+	Cortex A53	4	1	OP-TEE 3.8.0
Intel NUC 7BJYH	Pentium J5005	4	8	Intel SGX SDK v2.8
SiFive Unleashed	U540	4	8	Eyrie runtime v1.2.1

В таблице 3 приведены сравнение производительности для рассмотренных ДСИ. Время в ячейках таблицах указано в микросекундах и является усредненным значением 100 замеров.

Таблица 3 – Результаты сравнения производительности ДСИ

	П1	**П2**	**Чтение с диска**	**Запись на диск**
ARM TrustZone	26.39	297.32	229.44	24532.65
Intel SGX	22.30	53.49	15.73	9.20
Keystone	52.22	146.5	1377.72	1252.21

- первая колонка (П1) содержит в себе среднее время обращения к памяти последовательно;

- вторая колонка (П2) – среднее время обращения к памяти в случайном порядке;
- третья колонка – среднее время чтения данных с диска;
- четвертая колонка – среднее время записи на диск;
- пятая колонка – является решение аппаратным или программным.

Реализация ARM TrustZone имеет большую разницу по времени записи и чтения с диска. Это обусловлено особенностями реализации OP-TEE [16].

По результатам, приведенным в таблице 3, можно сделать вывод, что наилучшей производительностью обладает ДСИ Intel SGX. Это можно объяснить тем, что данная ДСИ построена с использованием аппаратных решениях компании Intel, спроектированных специальной для неё. Кроме того, в отличии от ARM TrustZone, ДСИ от компании Intel имеет более простую архитектуру с использованием анклавов.

1.4.4 Итоговая таблица

Результаты сравнений ДСИ по критериям приведены в таблице 4.

Таблица 4 – Сравнение реализаций ДСИ

	К1	**К2**	**К3**	**К4**	**К5**
ARM TrustZone	+	2	-	+	Аппаратное
Intel SGX	+	1	+	+	Аппаратное
Keystone	+	3	+	-	Программное

Из таблицы 4 можно сделать вывод, что каждая из рассмотренных реализаций ДСИ имеет свои достоинства и недостатки. Так, например, Intel SGX – наиболее «быстрая» доверенная среда исполнения (с точки зрения производительности), но является проприетарным решением от компании Intel. С другой стороны, Keystone является реализацией с полностью открытым исходным кодом, но проигрывает по производительности как ARM TrustZone, так и Intel SGX. Все ДСИ удовлетворяют критериям безопасности и имеют механизмы

защит от видов атак рассмотренных в данной работе.

1.5 Виртуализация в процессорных системах ARM

Виртуализация ARM позволяет существовать и выполняться более чем одной ОС в системе. Для этого используются специальные аппаратные решения и программное обеспечение для управления ими, которое называется гипервизор. С его помощью, несколько операционных систем могут разделять между друг другом один и тот же аппартный ресурс, например процессор или память. Главная роль гипервизора – распределение аппаратных ресурсов платформы и бесперебойная работа гостевых операционных систем с минимальными временными затратами.

Запущенная ОС не обладает информацией о других ОС, в результате чего для каждой из них создаётся иллюзия единоличного использования системы. Это стало возможно благодаря архитектурным особенностям виртуализации ARM, которые будут описаны ниже.

— Существует отдельный режим работы процессора EL2 1.2.1, который был создан специально для выполнения кода гипервизора.
— Поддержка маршрутизации исключений и виртуальных прерываний.
— Двухступенчатое преобразование адресов памяти: виртуальный адрес сначала транслируется в промежуточный (используемый гипервизором) и только потом в физический адрес. Это необходимо для изоляции гостевых ОС друг от друга.
— Отдельное исключение для перехода из режима EL1 в EL2 – Hypervisor Call (hvc).

На рисунке 10 представлена концептуальная схема системы с поддержкой виртуализации ARM. При загрузке системы запускается хостовая операционная система, которая инициализирует гипервизор. Он является компонентом ОС (чаще всего это модуль ядра), хоть и выполняется на другом уровне привилегий (EL2). Гипервизор отвечает за создание и распределение ресурсов между гостевыми и хостовой ОС. Как уже было описано ранее, все компоненты испол-

няются в обычном (не безопасном) мире.

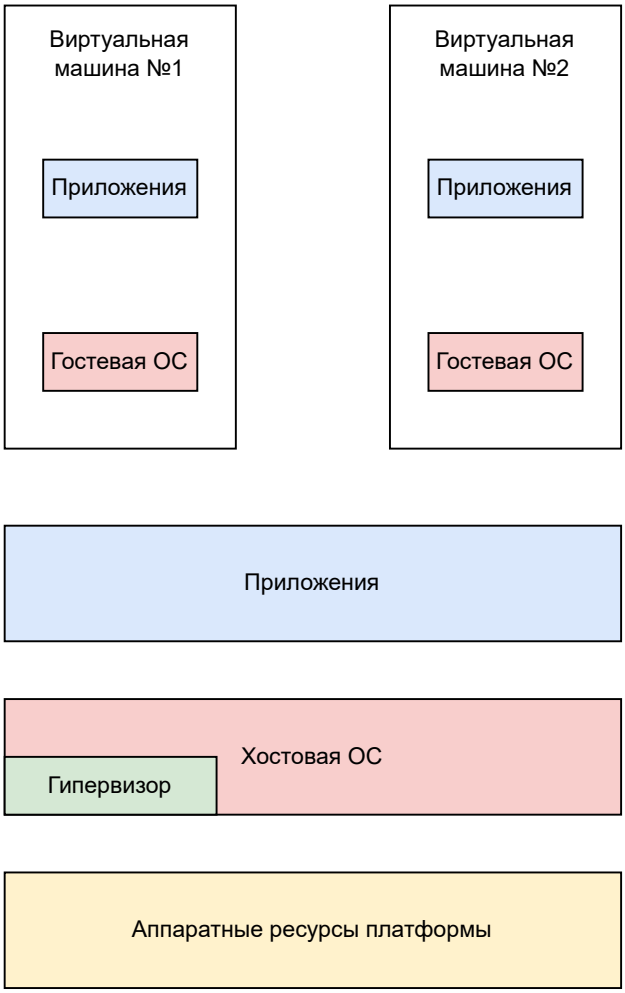

Рисунок 10 – Концептуальная схема системы с поддержкой виртуализации ARM.

1.5.1 Виртуализация ARM TrustZone

Технология ARM TrustZone не предназначена для виртуализации, из-за чего в виртуальной среде все виртуальные машины должны использовать одну и ту же доверенную среду исполнения. Такое решение не является эффективным и безопасным: найдя уязвимость в программном обеспечении ДСИ, злоумышленник получает доступ сразу ко всем виртуальным машинам [23].

1.6 Постановка задачи

Необходимо разработать и реализовать метод программной реализации ДСИ, который будет удовлетворять следующим условиям:

- метод должен работать на процессорах с архитектурой ARMv8 и новее;

- необходимо использовать аппаратную виртуализацию ARM;
- каждой виртуальной машине должна соответствовать одна независимая доверенная среда исполнения;
- метод должен поддерживать все свойства безопасности предоставляемые аппаратной поддерживаемой технология ARM TrustZone (описаны в разделе 1.2.1).

Вывод

В данном разделе представлен обзор следующих реализаций доверенных сред исполнения:

- ARM TrustZone;
- Intel SGX;
- Keystone.

Из которых только Keystone является программным решением с открытым исходным кодом, а первые два аппаратными проприетарными решениями. Сформулированы критерии оценки и проведено сравнение на их основе. Представлен обзор средств виртуализации в процессорных системах ARM. Представлена формализованная постановка задачи на разработку метода программной реализации доверенной среды исполнения с помощью виртуализации процессоров архитектуры ARM.

2 Конструкторский раздел

В конструкторском разделе разработан метод программной реализации доверенной среды исполнения с помощью виртуализации процессоров архитектуры ARM и представлено его формализованное описание в виде диаграмм IDEF0 и схем алгоритмов. Выполнено проектирование ПО для реализации данного метода.

2.1 Проектирование метода программной реализации доверенной среды исполнения

Разработанный метод предполагает использование принципа разделения функциональности на разные компоненты системы. Система спроектирована таким образом, что при атаке или получении доступа к одному из компонентов, её целостность нарушена не будет. Для корректной работы метода, система обязательно должна поддерживать технологию ARM TrustZone и аппаратные механизмы виртуализации ARM.

Каждой гостевой виртуальной машине сопоставляется виртуальная машина выполняющая роль доверенной среды исполнения, для достижения данной цели используется гипервизор. Каждая из этих виртуальных машин выполняется в обычном мире.

Для обеспечения целостности и проверки последовательности загрузки используется безопасный мир (который является частью ARM TrustZone). В безопасном мире располагаются модули отображения адресов памяти гипервизора и виртуальных машин; модуль перехода и сохранения контекста между виртуальными машинами. Такой подход обеспечивает целостность данных, даже если код гипервизора был скомпрометирован.

Каждая ДСИ использует своё индивидуальное рабочее окружение, в котором хранятся данные. Окружение закреплено за конкретной виртуальной машиной и доступное только когда процессор выполняется в режиме hypervisor, с помощью чего и добивается изоляция. При этом, само окружение не является

частью сущности гипервизора.

Для корректного и безопасного взаимодействия вышеописанных компонентов: модулей расположенных в безопасном мире, рабочих окружений и виртуальных машин используется модуль блокировки потока управления, который является частью гипервизора.

На рисунке 11 представлена диаграмма компонентов системы для разработанного метода программной реализации ДСИ.

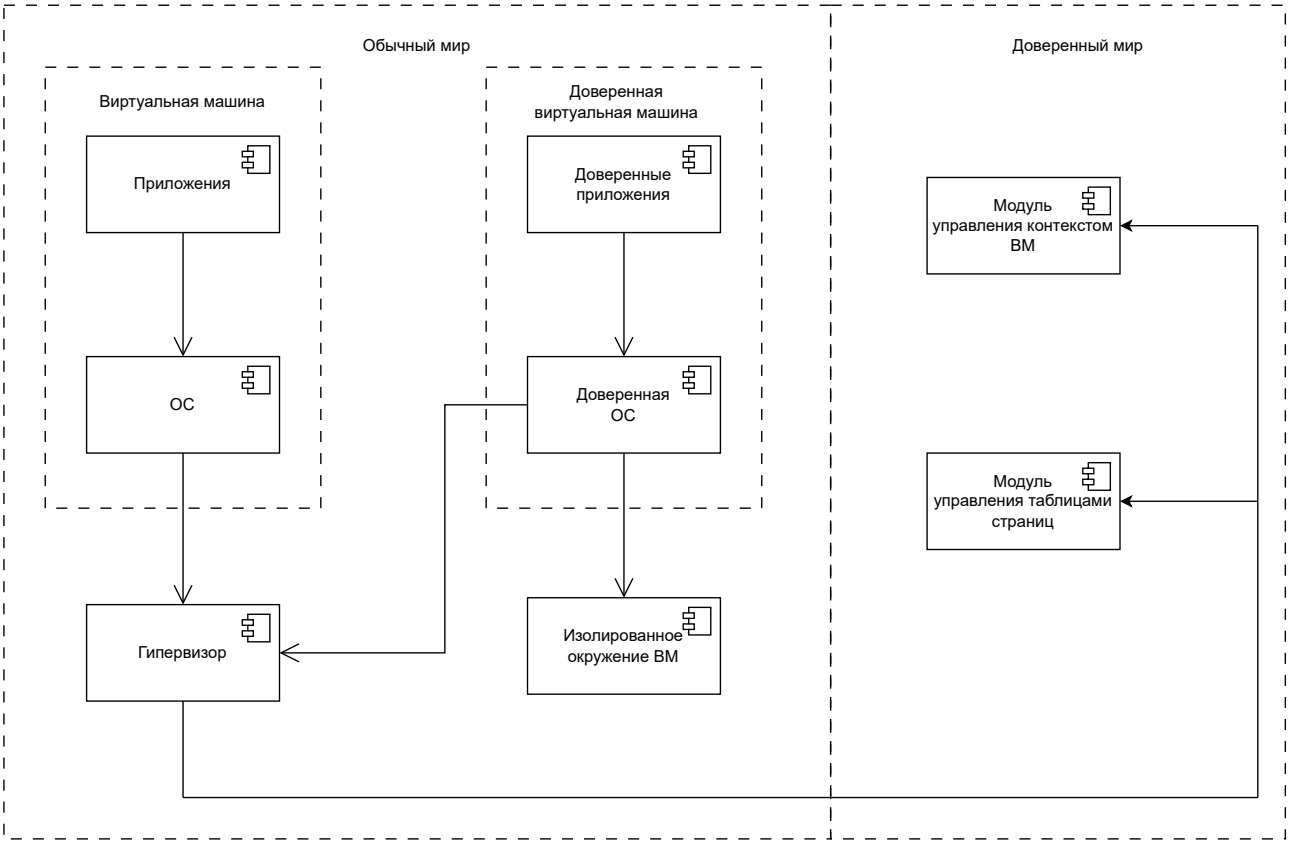

Рисунок 11 – Диаграмма компонентов системы

Далее будет дано детальное описание разрабатываемых модулей и окружений.

2.1.1 Модуль защищенного отображения памяти

Модуль отображения памяти отвечает за трансляцию виртуальных адресов в физические в режиме выполнения hypervisor, а так же промежуточных виртуальных в физические для гостевых виртуальных машин. Модуль выполняется в безопасном мире и предоставляет два интерфейса для гипервизора,

которые позволяют загрузить и модифицировать таблицу страниц.

Система построена таким образом, что модуль отображения памяти обладает монопольным доступом к загрузке и модификации таблицы страниц. Это достигается с помощью допущений, описанных ниже.

— Гипервизор не может изменять регистр, в котором хранится указатель на таблицу страниц: все инструкции, позволяющие это сделать, удаляются из его исходного кода. Сами страницы памяти, на которых располагаются таблицы страниц, помечаются как только для чтения для гипервизора.
— Страницы памяти, на которых расположен код гипервизора, помечаются как только для чтения. Это позволяет гарантировать что исходный код не может быть изменен во время исполнения.
— После запуска системы ни одна страница в адресном пространстве гипервизора не может быть помечена как исполняемая.

Таким образом, для внесения изменения в таблицу страниц необходимо использовать интерфейсы предоставляемые модулем отображения памяти, который управляет и реализует различные политики безопасности на каждое такое изменение.

2.1.2 Модуль блокировки потока управления

Чтобы принудительно передать управление на определенный код при возникновении какого-либо исключения используется модуль блокировки потока управления, который является частью гипервизора.

В архитектуре ARM, при возникновении исключения, управление передаётся специальному обработчику, адрес которого находится в таблице исключений. Код гипервизора модифицирован таким образом, что он лишён возможности модифицировать регистр содержащий базовый адрес таблицы, а так же её элементы. В обработчиках исключений, адреса которых указаны в этой таблице, добавлены специальные инструкции, которые гарантируют перенаправление потока управления к необходимым модулям (например, к модулю переключения контекста).

2.1.3 Модуль переключения контекста

Для переключения между контекстами виртуальных машин и гипервизора используется модуль, который выполняется в безопасном мире. Модуль отвечает за переключение между гостевой и доверенной виртуальной машиной и за переключения между виртуальными машинами и гипервизором. Оба типа переключений обрабатываются единообразно, так как в первом случае переключения так же обрабатывает гипервизор.

В архитектуре ARM существует две ситуации, которые могут привести к переключению выполнения из виртуальной машины в гипервизор:

1) аппаратное прерывание;
2) программное прерывание (вызов специальной инструкции процессора) или обработка исключительной ситуации.

В обоих случаях, переключение вызвано исключением, обработка которого будет произведена в данном модуле. Это гарантируется модулем блокировки потока управления.

Гипервизор может переключиться в контекст исполнения виртуальной машины изменив режим привилегий процессора из hypervisor (EL2) в kernel (EL1). Этого можно добиться тремя способами:

1) с помощью инструкции eret;
2) с помощью инструкции movs pc, lr;
3) явно установить режим привилегий.

Все данные вызовы в исходном коде гипервизора должны быть удалены и заменены на соответствующие вызовы предоставляемые модулем переключения контекста.

2.1.4 Индивидуальное рабочее окружение

За каждой доверенной виртуальной машиной закреплено индивидуальное рабочее окружение, выполняемое в режиме работы процессора hypervisor, которое эмулирует функциональность ARM TrustZone. Оно обладает своей таб-

лице страниц, стеком и данными.

Каждое окружение удовлетворяет следующим требованиям, которые позволяют его защитить, в случае если гипервизор был скомпрометирован:

— Единая точка входа.
— Запрещены прерывания. Код должен выполняться от точки входа до точки выхода.
— Нет зависимости от данных гипервизора.
— Данные окружения не передаются гипервизору.

Код каждого рабочего окружения загружается по фиксированному адресу в памяти, который задаётся на стадии компиляции, во время инициализации виртуальной машины. Модули, расположенные в безопасном мире, обладают информацией о метаданных каждого рабочего окружения (адрес точки входа и точки выхода). Сами страницы кода окружения помечаются как только для чтения. Первая и последняя выполняемая инструкция – это smc.

Входные данные (получаемые от виртуальной машины), доступны в режиме только для чтения. Для выходных данных выделяются отдельные страницы, так же доступные только для чтения. Чтобы записать в эти страницы какие-либо данные, необходимо сделать запрос к модулю отображения памяти.

Перед тем как передать управление коду рабочего окружения, страницы его стека настраиваются таким образом, что они доступны для чтения и записи только для ядра процессора, на котором сейчас выполняется его код. Так же, страницы его исходного кода помечаются как доступные для выполнения. Противоположные действия выполняются после исполнения последней инструкции рабочего окружения (smc). За эти действия отвечает модуль отображения памяти, находящийся в безопасном мире.

На рисунке 12 представлена диаграмма потоков данных для операции записи в хранилище рабочего окружения в нотации Йордона-Де Марко.

Рисунок 12 – Диаграмма потоков данных операции записи в хранилище рабочего окружения

2.2 Формализованное описание метода

Разрабатываемый метод – это виртуализация механизмов защиты, которые предоставляет аппаратная реализация ДСИ ARM TrustZone. На рисунках 13 - 14 представлена IDEF0-диаграмма и её детализированная версия метода программной реализации доверенной среды исполнения с помощью виртуализации процессоров архитектуры ARM. В следующих разделах будет представлено более подробное описание механизмов представленных на рисунке 14.

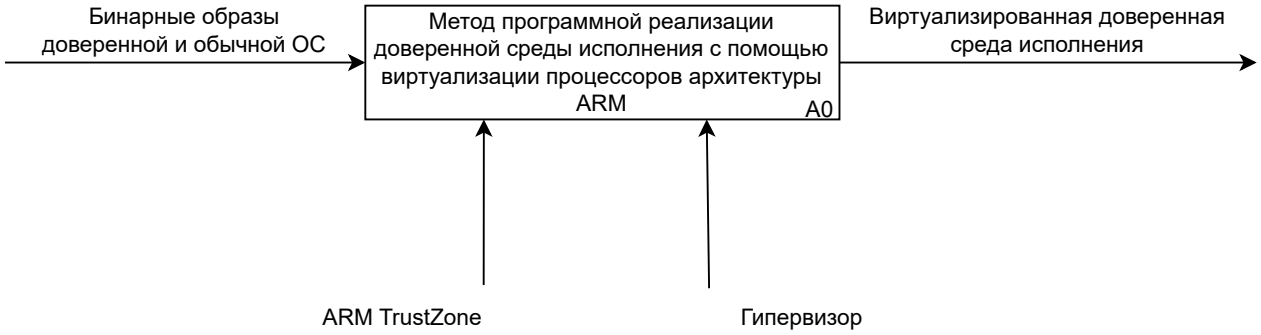

Рисунок 13 – IDEF0-диаграмма разработанного метода

2.2.1 Описание доверенной загрузки

Доверенная загрузка используется для обеспечения целостности загрузки системы. Процесс загрузки устройства с поддержкой ARM TrustZone включает в себя пять этапов.

— Загрузка загрузчика ОС из защищенного от воздействия внешних источников ПЗУ.

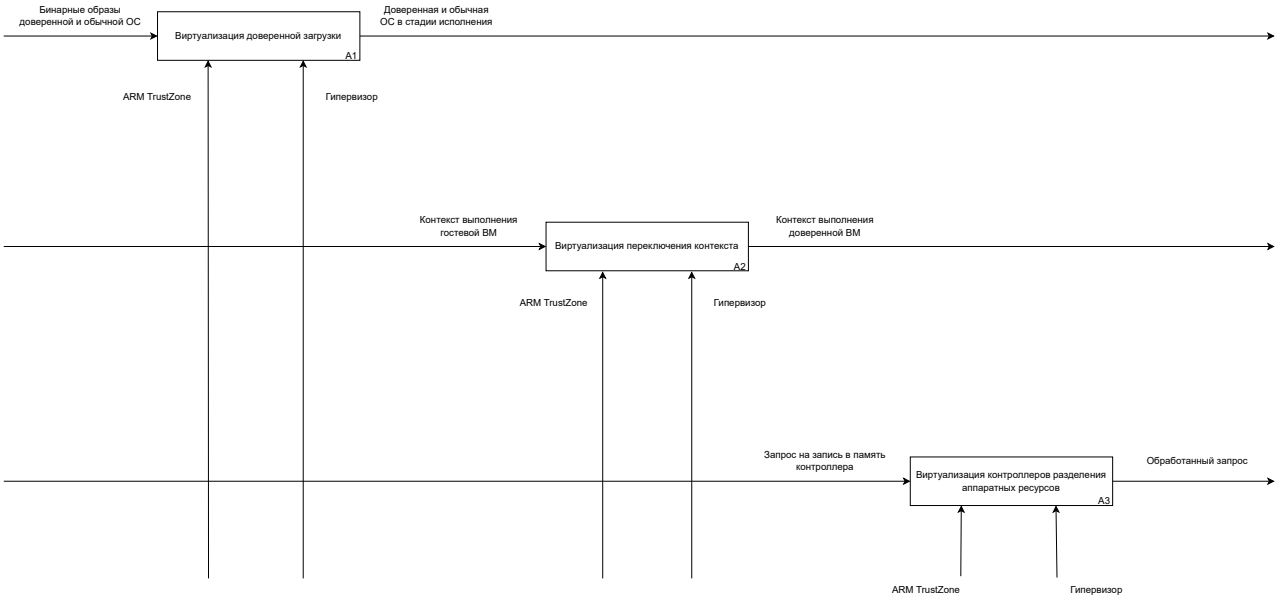

Рисунок 14 – Детализированная IDEF0-диаграмма разработанного метода

— Инициализация окружения и загрузка ядра доверенной ОС.
— Доверенное ядро инициализирует и настраивает окружение для корректной работы безопасного мира.
— Доверенное ядро загружает в память загрузчик обычной ОС и передаёт ему управление.
— Загрузчик обычной ОС вычисляет контрольную сумму бинарного образа ОС перед его загрузкой в память и загружает ОС.

Для виртуализации доверенной загрузки необходимо обеспечить следующие свойства:

1) ядро доверенной ОС должно загрузиться до загрузки обычного;
2) образ обычной ОС должен быть проверен;
3) образ ОС не может быть подменён.

На рисунке 15 представлена детализированная IDEF0-диаграмма виртуализации доверенной загрузки. Данная схема полностью удовлетворяет свойствам, которые были описаны выше.

На рисунке 16 представлена схема алгоритма регистрации виртуальных машин в безопасном мире (в модуле переключения контекста).

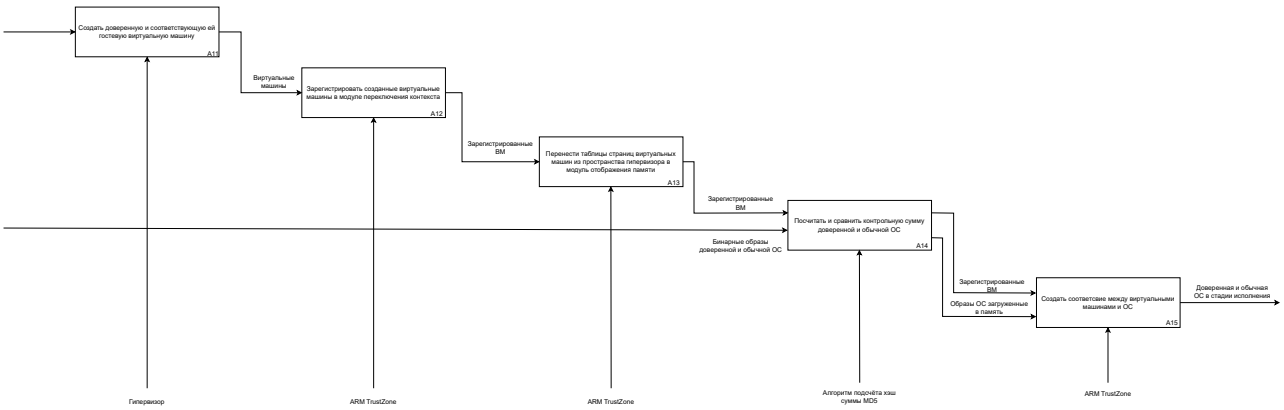

Рисунок 15 – Детализированная IDEF0-диаграмма виртуализации доверенной загрузки

2.2.2 Описание защиты и переключение контекста выполнения

Каждый из миров (безопасный и обычный) имеют свой контекст выполнения – значения регистров, настройка памяти, структуры данных и так далее. Реализация ARM TrustZone предоставляет функциональности защиты этого контекста – каждый контекст доступен для чтения и записи только из мира, к которому он принадлежит.

На рисунке 17 представлена детализированная IDEF0-диаграмма виртуализации переключения контекста между гостевой виртуальной машиной и доверенной. Контекст выполнения остаётся защищенным, т.к. сохраняется и восстанавливается он из безопасного мира. Обратная схема переключения контекста (из доверенной в гостевую ВМ) аналогична.

На рисунке 18 представлена схема алгоритма сохранения контекста выполнения виртуальной машины в безопасном мире.

2.2.3 Описание разделения аппаратных ресурсов

ARM TrustZone разделяет аппаратные ресурсы (память, периферия, прерывания) между безопасным и обычном миром. Для каждого из этих ресурсов существует отдельный контроллер, который отвечает за их настройку и распределение между мирами. Все три контроллера могут быть настроены только из безопасного мира. Таким образом, необходимо виртуализировать каждый их

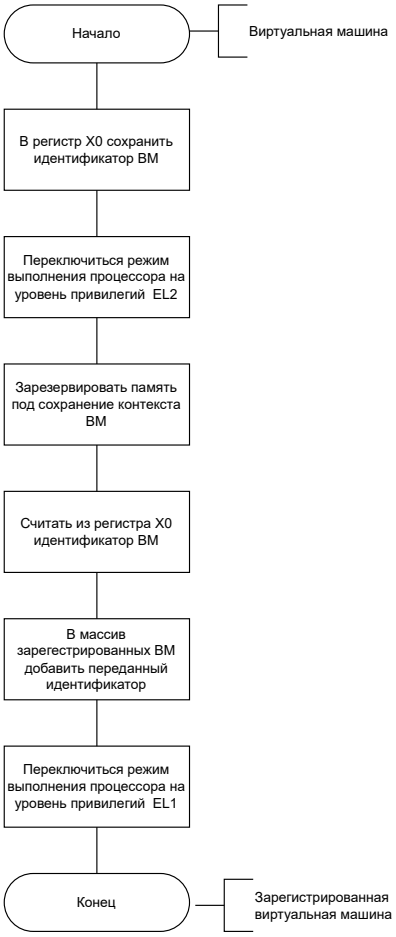

Рисунок 16 – Схема алгоритма регистрации виртуальных машин в безопасном мире

этих контроллеров.

Для каждой виртуальной машины область памяти в которой находятся контроллеры помечена как только для чтения. В результате попытки записи в эти контроллеры, процессор вызывает исключение и управление передаётся гипервизору для его дальнейшей обработки. Такой подход называется trap-and-emulate [22].

На рисунке 19 представлена детализированная IDEF0-диаграмма виртуализации контроллеров разделения аппаратных ресурсов.

Вывод

Был разработан метод программной реализации доверенной среды исполнения с помощью виртуализации процессоров архитектуры ARM. Были спроектированы и разработаны следующие компоненты:

Рисунок 17 – Детализированная IDEF0-диаграмма виртуализации переключения контекста

- модуль защищенного отображения памяти;
- модуль блокировки потока управления;
- модуль переключения контекста;
- индивидуальное рабочее окружение.

Представлено формализованное описание метода в виде IDEF0-диаграмм, состоящей из трех уровней, которые так же были детализированы:

- виртуализация доверенной загрузки;
- виртуализация переключения контекста;
- виртуализация контроллеров разделения аппаратных ресурсов.

С помощью схем алгоритмов описаны используемые в разработанном методе алгоритмы:

- регистрация виртуальных машин в безопасном мире;
- сохранение контекста выполнения виртуальной машины.

Рисунок 18 – Схема алгоритма контекста выполнения виртуальной машины

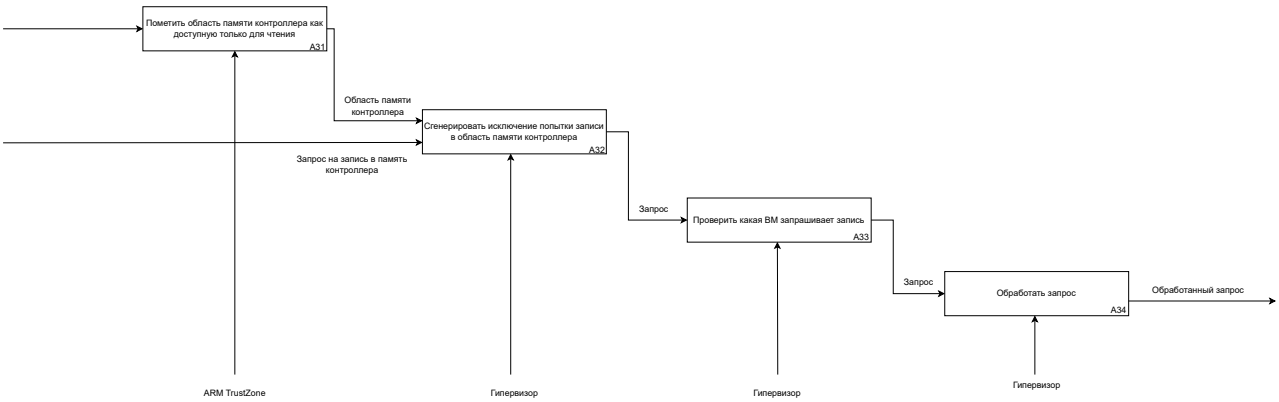

Рисунок 19 – Детализированная IDEF0-диаграмма виртуализации контроллеров разделения аппаратных ресурсов

3 Технологический раздел

В данном разделе обоснован выбор средства программной реализации доверенной среды исполнения с помощью виртуализации процессоров архитектуры ARM. Дано описание ключевого функционала, реализующее метод доверенной среды исполнения с помощью виртуализации процессоров архитектуры ARM.

3.1 Выбор операционной системы

В качестве гостевой операционной системы была выбрана ОС Linux с версией ядра 6.17 [20]. Выбор обоснован тем, что Linux является полностью совместимым с технологий ARM TrustZone, а так же обладает открытым исходным кодом.

В качестве доверенной операционной системы была выбрана ОС OP-TEE версии 4.0. Выбор основан тем, что эта ОС обладает большим объемом документации и открытым исходным кодом.

3.2 Выбор средств виртуализации

В качестве гипервизора был выбран KVM (Kernel-Based Virtual Machine) [21], который является частью ядра ОС Linux. Данный выбор обоснован тем, что KVM является технологией с открытым исходным кодом, полностью поддерживается для процессоров архитектуры ARM и является совместимым с ОС Linux.

3.3 Сборка программного обеспечения

Разработанное программное обеспечение является модификацией ядра OP-TEE. Для сборки проектов используется специальная утилита make [**?**], позволяющая автоматизировать сборку ядра. make является кроссплатформенной системой автоматизации сборки программного обеспечения из исходного кода. make позволяет существенно ускорить процесс сборки проекта. Так, например, при изменении одного исходного файла проекта, заново будет собран в объектный файл лишь этот исходный файл, а не все файлы проекта. В листинге 1

представлен скрипт для сборки ядра OP-TEE.

Листинг 1: Сборочный скрипт OP-TEE

```
#!/bin/sh

mkdir build
cd build
make toolchains
make
```

В рамках данной работы так же была произведена модификация гипервизора KVM. KVM собирается в составе ядра Linux, поэтому для его сборки необходимо включить опцию `CONFIG_KVM` в defconfig ядра. Так же для сборки используется утилита make. В листинге 2 приведен скрипт для сборки ядра Linux вместе с KVM.

Листинг 2: Сборочный скрипт ядра Linux вместе с KVM

```
make mrproper
ARCH=arm64 make defconfig
ARCH=arm64 make
```

3.4 Требования к вычислительной системе

Для сборки и установки разработанного программного обеспечения требуются следующие библиотеки и утилиты, представленные в таблице 5.

3.5 Структура программного обеспечения

Разработанное ПО представляет из себя модификации исходного кода гипервизора KVM и ОС OP-TEE. Ниже описан ключевой функционал, необходимый для работы системы.

Таблица 5 – Таблица ПО, необходимого для сборки разработанного программного обеспечения

ПО	**Минимальная версия**
gcc	10.2
GNU make	3.80
binutils	2.12
util-linux	2.10o
module-init-tools	0.9.10
e2fsprogs	1.41.4
jfsutils	1.1.3
reiserfsprogs	3.6.3
xfsprogs	2.6.0
squashfs-tools	4.0
btrfs-progs	0.18
pcmciautils	004
quota-tools	3.09
PPP	2.4.0
isdn4k-utils	3.1pre1
nfs-utils	1.0.5
procps	3.2.0
oprofile	0.9
udev	081
grub	0.93
mcelog	0.6
iptables	1.4.2
openssl, libcrypto	1.0.0
bc	1.2

3.5.1 Функция блокировки потока управления

При возникновении какого-либо исключения используется функция блокировки потока управления (часть гипервизора KVM). Код необходимых обработчиков исключений был модифицирован таким образом, чтобы управление передавалось необходимым модулям. В листинге 3 приведен модифицированный код обработчика исключений.

Листинг 3: Модификация кода обработчика исключений

```
static void enter_exception64(struct kvm_vcpu *vcpu,
                  unsigned long target_mode,
                  enum exception_type type)
{
    unsigned long sctlr, vbar, old, new, mode;
    u64 exc_offset;

    mode = *vcpu_cpsr(vcpu) & (PSR_MODE_MASK | PSR_MODE32_BIT);

    if (mode == target_mode)
        exc_offset = CURRENT_EL_SP_ELx_VECTOR;
    else if ((mode | PSR_MODE_THREAD_BIT) == target_mode)
        exc_offset = CURRENT_EL_SP_EL0_VECTOR;
    else if (!(mode & PSR_MODE32_BIT))
        exc_offset = LOWER_EL_AArch64_VECTOR;
    else
        exc_offset = LOWER_EL_AArch32_VECTOR;

    switch (target_mode) {
    case PSR_MODE_EL1h:
        vbar = __vcpu_read_sys_reg(vcpu, VBAR_EL1);
        sctlr = __vcpu_read_sys_reg(vcpu, SCTLR_EL1);
        __vcpu_write_sys_reg(vcpu, *vcpu_pc(vcpu), ELR_EL1);
        break;
    case PSR_MODE_EL2h:
```

```
26              vbar = __vcpu_read_sys_reg(vcpu, VBAR_EL2);
27              sctlr = __vcpu_read_sys_reg(vcpu, SCTLR_EL2);
28              __vcpu_write_sys_reg(vcpu, *vcpu_pc(vcpu), ELR_EL2);
29              break;
30          default:
31              BUG();
32          }
33
34          *vcpu_pc(vcpu) = vbar + exc_offset + type;
35
36          old = *vcpu_cpsr(vcpu);
37
38          if (need_intercepted(old)) {
39              new = do_optee_handle_action(target_mode, type, old);
40          } else {
41              new = 0;
42
43              new |= (old & PSR_N_BIT);
44              new |= (old & PSR_Z_BIT);
45              new |= (old & PSR_C_BIT);
46              new |= (old & PSR_V_BIT);
47
48              if (kvm_has_mte(kern_hyp_va(vcpu->kvm)))
49                  new |= PSR_TCO_BIT;
50
51              new |= (old & PSR_DIT_BIT);
52
53              new |= (old & PSR_PAN_BIT);
54              if (!(sctlr & SCTLR_EL1_SPAN))
55                  new |= PSR_PAN_BIT;
56
57              if (sctlr & SCTLR_ELx_DSSBS)
58                  new |= PSR_SSBS_BIT;
59
60              new |= PSR_D_BIT;
```

```
61              new |= PSR_A_BIT;
62              new |= PSR_I_BIT;
63              new |= PSR_F_BIT;
64
65              new |= target_mode;
66          }
67
68          *vcpu_cpsr(vcpu) = new;
69          __vcpu_write_spsr(vcpu, target_mode, old);
70      }
```

3.5.2 Функция переключения контекста

Переключение контекста между ВМ реализуется в модуле переключения контекста, который исполняется в доверенном мире. Этот модуль предоставляет API, позволяющее переключить контекст из гостевой ВМ в доверенную и наоборот. В листинге 4 представлен исходный код этой функции.

Листинг 4: Переключение контекста между двумя виртуальными машинами

```
1   static int __swap_to_next_vm(vm_id_t vm_id, unsigned flags)
2   {
3       struct secchg_partition *prtn;
4       uint32_t exceptions;
5
6       prtn = get_current_prtn();
7
8       if (prtn && prtn->id == vm_id)
9           return 0;
10
11      exceptions = cpu_spin_lock_xsave(&prtn_list_lock);
12      LIST_FOREACH(prtn, &prtn_list, link) {
13          if (prtn->id == guest_id) {
14              set_current_prtn(prtn, flags);
```

```
15                  core_mmu_set_prtn(prtn->mmu_prtn, flags);
16                  refcount_inc(&prtn->refc);
17                  cpu_spin_unlock_xrestore(&prtn_list_lock,
18                                           exceptions);
19                  return 0;
20              }
21          }
22          cpu_spin_unlock_xrestore(&prtn_list_lock, exceptions);
23
24          return 0;
25      }
26
27      int secchg_change_context(vm_id_t vm_id, unsigned flags)
28      {
29          struct secchg_vm_map current =
30              secchg_get_current_vm_map();
31          struct secchg_vm_map swap_to =
32              secchg_get_vm_partition(vm_id);
33          int ret;
34
35          if (secchg_validate_vm_map(swap_to)) {
36              DMSG("failed to change context to %d", vm_id);
37              return -EINVAL;
38          }
39
40          ret = __save_vm_map(current);
41          if (ret)
42              return ret;
43
44          __save_current_context();
45
46          return __swap_to_next_vm(vm_id, flags);
47      }
```

3.5.3 Функция защищенного отображения памяти

Модуль отображения памяти, который так же исполняется в контексте доверенной ОС, отвечает за функции защищенного отображения памяти между виртуальными машинами. В листинге 5 представлена функция, позволяющая обходить страницы памяти ВМ и в зависимости от входных параметров выполнять действия над ними.

Листинг 5: Отображение памяти, реализованное на уровне доверенной ОС

```
static inline int __sec_vm_pgtable_visit(
    struct sec_vm_pgtable_walk_data *data,
    struct sec_vm_pgtable_mm_ops *mm_ops,
    sec_vm_pteref_t pteref, s8 level)
{
    enum sec_vm_pgtable_walk_flags flags = data->walker->flags;
    sec_vm_pte_t *ptep = data->walker->ptep;
    struct sec_vm_pgtable_visit_ctx ctx = {
        .ptep    = ptep,
        .old    = READ_ONCE(*ptep),
        .arg    = data->walker->arg,
        .mm_ops    = mm_ops,
        .start    = data->start,
        .addr    = data->addr,
        .end    = data->end,
        .level    = level,
        .flags    = flags,
    };
    int ret = 0;
    bool reload = false;
    sec_vm_pteref_t childp;
    bool table = sec_vm_pte_table(ctx.old, level);

    if (table && (ctx.flags & sec_vm_PGTABLE_WALK_TABLE_PRE)) {
        ret = sec_vm_pgtable_visitor_cb(data, &ctx,
```

```
                    sec_vm_PGTABLE_WALK_TABLE_PRE);
            reload = true;
        }

        if (!table && (ctx.flags & sec_vm_PGTABLE_WALK_LEAF)) {
            ret = sec_vm_pgtable_visitor_cb(data, &ctx,
                    sec_vm_PGTABLE_WALK_LEAF);
            reload = true;
        }

        if (reload) {
            ctx.old = READ_ONCE(*ptep);
            table = sec_vm_pte_table(ctx.old, level);
        }

        if (!sec_vm_pgtable_walk_continue(data->walker, ret))
            goto out;

        if (!table) {
            data->addr = ALIGN_DOWN(data->addr,
                    sec_vm_granule_size(level));
            data->addr += sec_vm_granule_size(level);
            goto out;
        }

        childp = (sec_vm_pteref_t)
            sec_vm_pte_follow(ctx.old, mm_ops);
        ret = __sec_vm_pgtable_walk(data, mm_ops, childp,
                    level + 1);
        if (!sec_vm_pgtable_walk_continue(data->walker, ret))
            goto out;

        if (ctx.flags & sec_vm_PGTABLE_WALK_TABLE_POST)
            ret = sec_vm_pgtable_visitor_cb(data, &ctx,
                    sec_vm_PGTABLE_WALK_TABLE_POST);
```

```
61
62  out:
63      if (sec_vm_pgtable_walk_continue(data->walker, ret))
64          return 0;
65
66      return ret;
67  }
68
69  static int sec_vm_pgtable_walk(
70      struct sec_vm_pgtable_walk_data *data,
71      struct sec_vm_pgtable_mm_ops *mm_ops,
72      sec_vm_pteref_t pgtable, s8 level)
73  {
74      u32 idx;
75      int ret = 0;
76
77      if (level < sec_vm_PGTABLE_FIRST_LEVEL ||
78              level > sec_vm_PGTABLE_LAST_LEVEL)
79          return -EINVAL;
80
81      for (idx = sec_vm_pgtable_idx(data, level);
82          idx < PTRS_PER_PTE; ++idx) {
83          sec_vm_pteref_t pteref = &pgtable[idx];
84
85          if (data->addr >= data->end)
86              break;
87
88          ret = __sec_vm_pgtable_visit(data, mm_ops,
89                          pteref, level);
90          if (ret)
91              break;
92      }
93
94      return ret;
```

```
95      }
```

Вывод

В данном разделе были описаны средства разработки программного обеспечения, требования к ПО и системе для сборки этого ПО. Было приведено описание ключевых функций разработанного метода:

- функция блокировки потока управления, реализованная на уровне KVM;
- функция переключения контекста между ВМ;
- функция защищенного отображения памяти ВМ.

4 Исследовательский раздел

В данном разделе проведено исследование эффективности и применимости разработанного программного обеспечения. Выполнено сравнение результатов работы разработанного метода и метода с аппаратной поддержкой доверенной среды исполнения на базе процессоров с архитектурой ARM (ARM TrustZone).

4.1 Методика проведения исследования

Для того чтобы иследовать эффективность и применимость разработанного программного обеспечения, необходимо сравнить количество машинных инструкций для выполнения задач ДСИ: смена контекста между мирами, проверка целостности, обработку прерываний и так далее. Для точного подсчета количества выполняемых инструкций за промежуток времени, был использовано аппаратное расширение ARM Performance Monitoring Unit [24]. Было подсчитано количество инструкций для выполнения одинаковых задач с использованием виртуализации ARM TrustZone и аппаратного решения.

В качестве ядра гостевой ОС был использовано ядро Linux версии 6.15, а в качестве ядра привилегированной ОС – OP-TEE версии 4.0. В качестве гипервизора был использован KVM, который является частью ядра Linux.

Сравнение было проведено как для 32-битных процессоров с архитектурой ARMv7 так и для более новых, 64-битных процессоров с архитектурой ARMv8. БЫли выбраны два устройства: Raspberry Pi 4 Model B [25] и Raspberry Pi 2 Model [26]. Raspberry Pi 4 Model B обладает следующими характеристиками:

— Четыре 64-битных ядра Cortex A72 (ARMv8) с тактовой частотой 1,5Ггц.
— 8 Гб ОЗУ.

Raspberry Pi 2 обладает следующими характеристиками:

— Четыре 32-битных ядра Cortex A7 (ARMv7) с тактовой частотой 0.9ГГц.
— 1 Гб ОЗУ.

Во время проведения исследования для каждой виртуальной машины было выделен 1 виртуальный CPU который соотвествует 1 физическому CPU.

4.2 Сравнение количества машинных инструкций с аппаратной реализацией

4.2.1 Сравнение при выполнение ключевых задач

Можно выделить три ключевые задачи, выполняемых доверенной средой исполнения, без которых она не может считаться полноценной:

1) смена контекста выполнения между гостевым и доверенным миром;
2) разделение аппаратных ресурсов;
3) проверка целостности загружаемых образов ОС.

В таблице 6 представлено сравнение количества машинных инструкций для смены контекста выполнения и разделения аппаратных ресурсов между мирами: разделение памяти и прерываний. В первых двух столбцах указаны результаты сравнения на устройстве Raspberry Pi 2 Model B; первый столбец – аппаратная реализация, второй – визуализация (разработанный метод). В третьем и четвёртом столбце указанные результаты сравнения выполняемые на устройстве Raspberry Pi 4 Model, для аппаратной и виртуализированной реализации соответственно.

Таблица 6 – Сравнение количества инструкций необходимых для выполнения ключевых задач ДСИ

	R Pi2 (A)	**R Pi2 (B)**	**R Pi4 (A)**	**R Pi4 (B)**
Смена контекста	9200	18745	1525	7625
Разделение участков памяти	3245	7055	2208	7800
Разделение прерываний	2273	6251	986	3421

По результатам из таблицы 6 можно сделать вывод, что смена контекста для 32-битных процессоров в разработанном методе требует в два раза больше инструкций, чем в аппаратной реализации ARM TrustZone. Для 64-битных

процессоров разница составляет порядка 5 раз. Данную разницу можно счесть приемлемой, так как смена контекста между мирами происходит редко и практически никак не отражается на производительности выполняемых приложений и всей системы в целом.

Для выполнения разделения участков памяти, по сравнению с аппаратной реализацией, необходимо в 2 и 4 раза для 32-битных и 64-битных процессоров соответственно. Для разделения и корректной маршрутизации прерываний необходимо в 3 раза больше инструкций как для 32-битных, так и для 64-битных процессоров. Данную разницу, так же, как и в случае со сменой контекста, можно счесть приемлемой, так как операции разделения ресурсов выполняются редко.

В таблице 7 представлено сравнение количества машинных инструкций для проверки целостности загружаемых образов ОС. Для этого было проведено хеширование регионов памяти размером 1, 16, 32, 64 и 128 Кб с помощью алгоритма SHA256 [27].

Таблица 7 – Сравнение количества инструкций необходимых для выполнения проверки целостности

	R Pi2 (A)	**R Pi2 (B)**	**R Pi4 (A)**	**R Pi4 (B)**
1 Кб	1150	1256	300	325
16 Кб	17890	19200	4450	4800
32 Кб	36502	38600	10200	11223
64 Кб	80215	84555	22254	23507
128 Кб	165865	170205	50700	51998

По результатам из таблицы 7, можно сделать вывод что накладные расходы для выполнения проверки целостности в среднем составляют 5-10% по сравнению с аппаратной реализацией, что не является критичным и не влияет на производительность системы.

4.2.2 Сравнение с использованием пользовательских приложений

Для сравнения накладных ресурсов при использовании пользовательских приложений были выбраны приложения ccrypt и GoHttp. ccrypt был использован для шифрования файлов размеров 1 Кб, а GoHttp для передачи по сети. Логика шифрования данных реализована на уровне доверенной среды исполнения.

Было проведено сравнение количества выполняемых как и с одной парой (гостевая и доверенная ОС) виртуальных машин, так и при запуске нескольких пар одновременно.

На рисунках 20 и 21 представлено сравнение (в процентах) количества используемых инструкций при использовании одной пары ВМ. Аппаратная реализция отмечена как 100%.

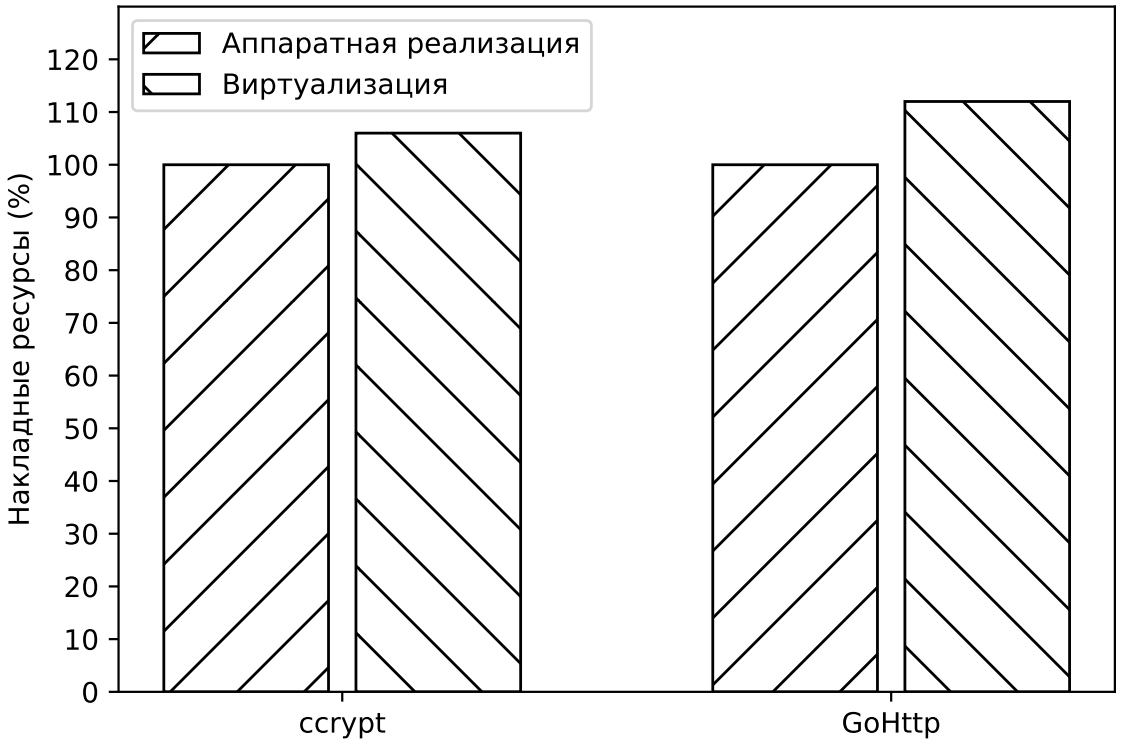

Рисунок 20 – Сравнение результатов выполнения пользовательских приложений (ARMv7)

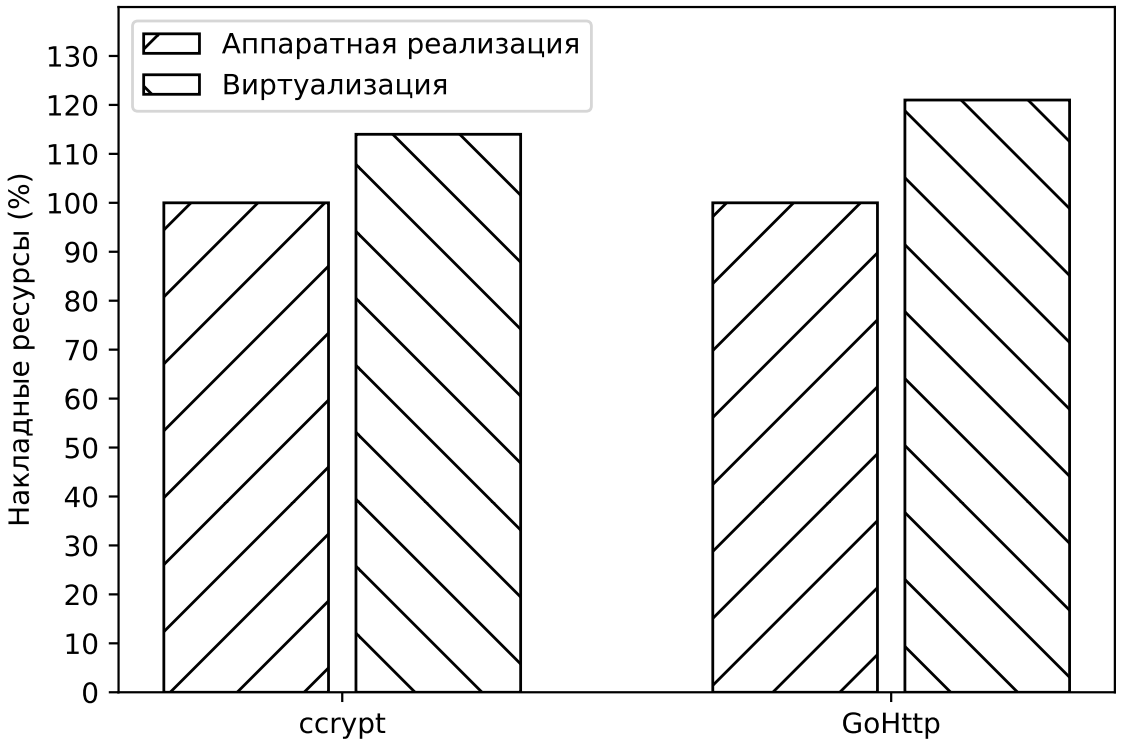

Рисунок 21 – Сравнение результатов выполнения пользовательских приложений (ARMv8)

На рисунках 22 и 23 представлено сравнение количества инструкций при использовании нескольких пар ВМ, которые передают файлы размером 1Кб между друг другом с использованием приложения GoHTTP.

Можно сделать вывод, что при использовании одной пары ВМ, накладные ресурсы на исполнение пользовательских приложений в среднем составляют 7-15%. При использовании нескольких пар ВМ, накладные ресурсы возрастают и составляют от 20 до 50% по сравнению с аппаратной реализацией. Заметный рост накладных ресурсов наблюдается при использовании двух пар ВМ (20% и 35% для ARMv7 и ARMv8 соответственно), но незначительный рост в 5-10% при увеличении количества пар (две и более). Можно сделать вывод, что ключевую роль в увеличении накладных расходов играет тот факт, что параллельно используется более одной пары ВМ, а не зависит от их количества: разница при

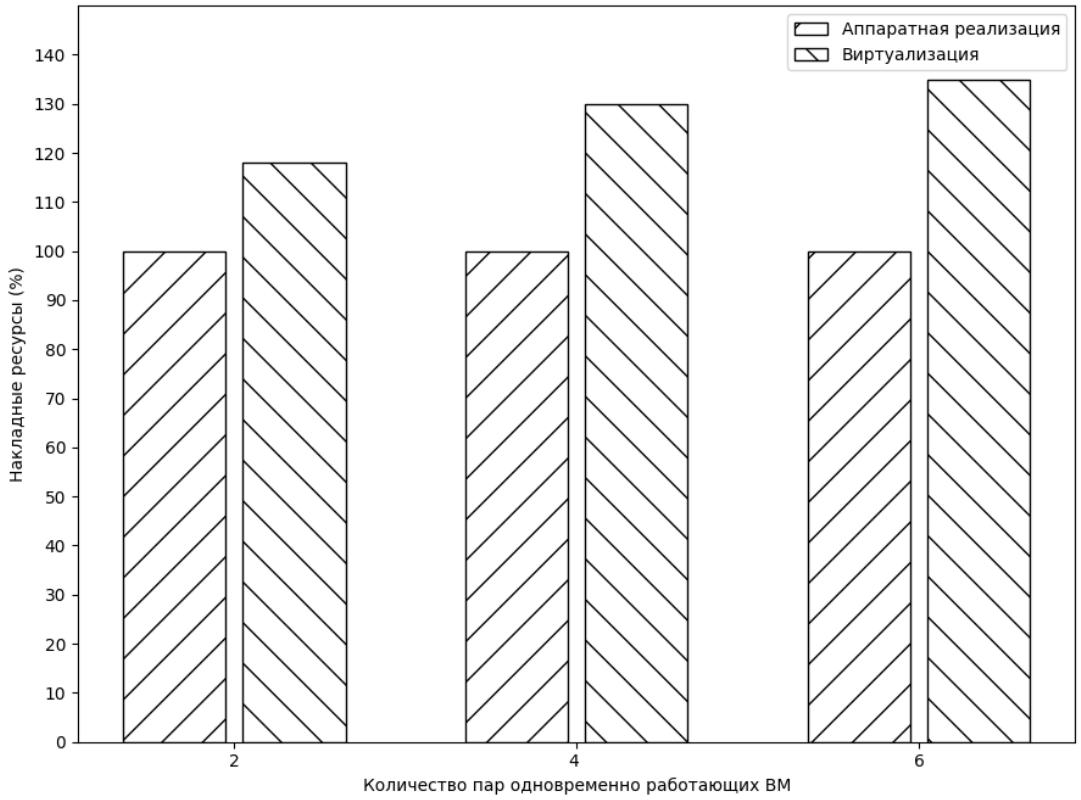

Рисунок 22 – Сравнение результатов выполнения пользовательских приложений (ARMv7), несколько пар ВМ

использовании 2 и 6 пара ВМ составляет 7-15%.

4.2.3 Сравнение с использованием серверных приложений

Для тестирования накладных ресурсов при использовании серверных приложений были выбраны MongoDB [28] и Apache [29]. Количество виртуальных CPU для каждой ВМ было увеличено до 4. Используется одна пара ВМ. В качестве устройства использовался Raspberry Pi 4 Model B (ARMv8).

На рисунке 24 представлены результаты сравнения с использованием MongoDB: клиент, расположенный на той же ВМ, что и сервер, на протяжении 1 минуты вставляет объекты различного размера в таблицу базы данных.

Из риснука 24 можно сделать вывод о том, что при использовании разработанного метода скорость записи в сервер MongoDB практически идентична

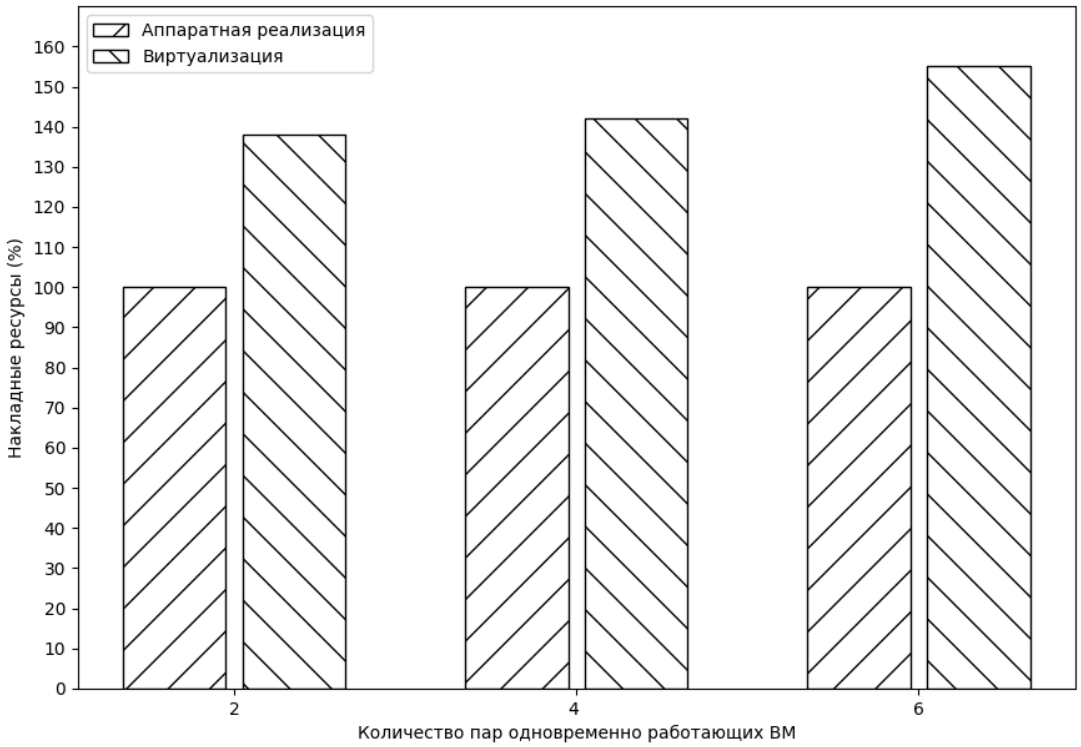

Рисунок 23 – Сравнение результатов выполнения пользовательских приложений (ARMv8), несколько пар ВМ

скорости записи при использовании аппаратного метода: разница составляет 5-10%.

На рисунке 25 представлены результаты сравнения с использованием Apache: клиент, расположенный на той же ВМ, что и сервер, на протяжении 1 минуты скачивает файл различного размера с сервера.

Из рисунка 24 можно так же сделать вывод, что скорость чтения с сервера Apache при использовании разработанного метода близка к скорости с использованием аппаратной реализации: разница составляет 10-15%.

Вывод

В данном разделе проведено исследование эффективности разработанного программного обеспечения. Было произведено сравнение количества используемых машинных инструкций для выполнения ключевых задач ДСИ:

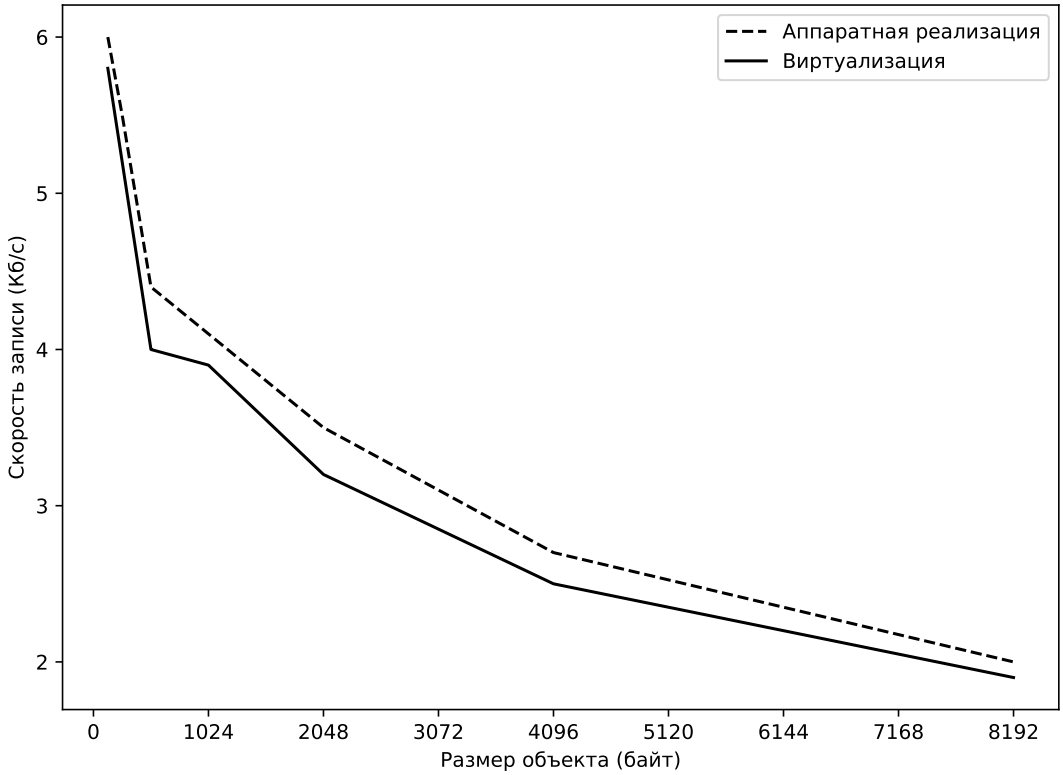

Рисунок 24 – Зависимость скорости записи на сервер MongoDB от размера объекта

— Для смены контекста требуется в 2 и 5 раз больше инструкций для 32-битных (ARMv7) и 64-битных (ARMv8) процессоров соотвественно.
— Для обработки разделения аппаратных ресурсов в среднем требуется в 2-4 раза больше инструкций.
— Разница в количестве инструкций для проверки целостности образов ОС между разработанным методом и аппаратной реализацией в среднем составляет 5-10%.

Во всех приведенных сравнениях, для 32-битных процессоров разница в количестве инструкций составляет меньше, чем для 64-битных (в среднем в 1.5-2 раза).

Произведено сравнение накладных ресурсов при использовании пользовательских и серверных приложений:

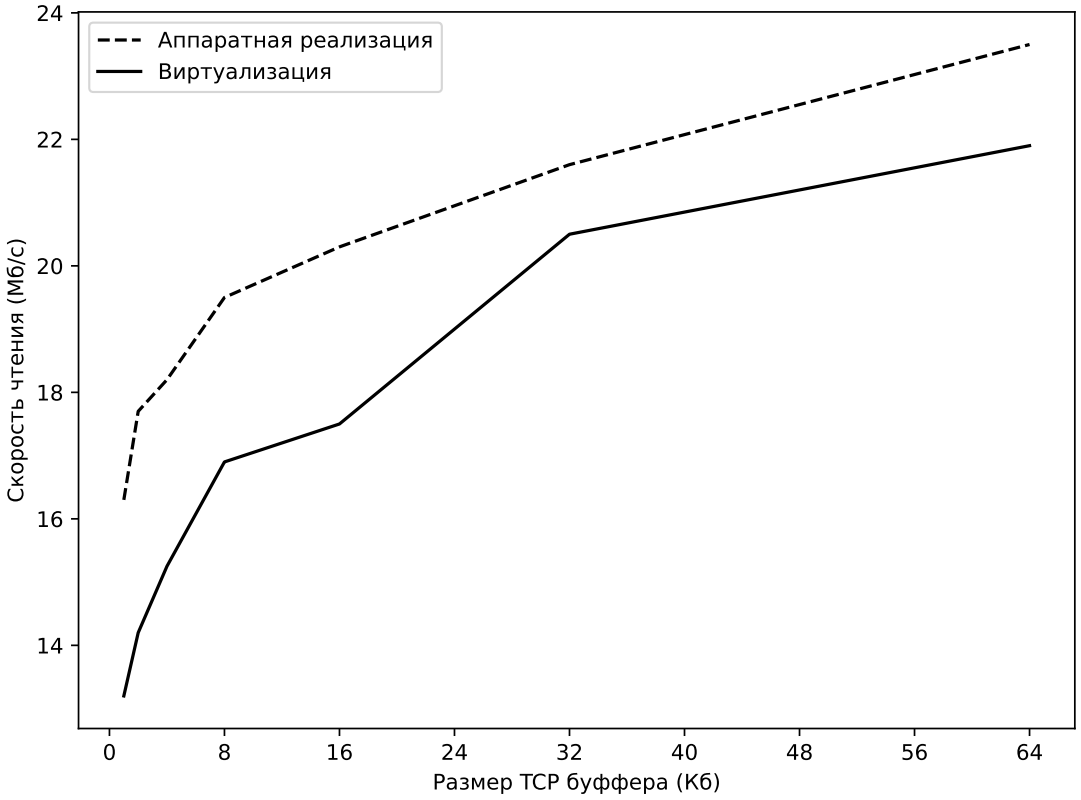

Рисунок 25 – Зависимость скорости чтения данных с сервера Apache от размера TCP буффера

- Для пользовательских приложений разница с аппаратной реализацией составляет 7-15% и 20-50% при использовании нескольких виртуальных машин одновременно.
- Накладные расходы резко возрастают (на 20-40%) если в системе используется более одной пары ВМ.
- Произведено сравнение скорости записи данных в сервер MongoDB: разница с аппаратной реализацией составляет 5-10 в зависимости от размера данных%.
- При чтении файлов с сервера Apache было установлено, что скорость чтения на 10-15% меньше, чем при использовании аппаратной реализации.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы была достигнута ее цель – изучены существующие реализации доверенных сред исполнения и разработан метод программной реализации доверенной среды исполнения с помощью виртуализации процессоров архитектуры ARM.

Для достижения поставленной цели были решены следующие задачи:

- проведен обзор существующих реализаций ДСИ;
- описаны их достоинства и недостатки;
- сформулированы критерии сравнения и сравнены реализации;
- изложены особенности метода;
- представлена формализация в виде диаграмм IDEF0 и схем алгоритмов;
- выполнено проектирование ПО для реализации метода;
- обосновано выбор средств программной реализации;
- реализовано ПО;
- проведено исследование эффективности и применимости ПО.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Introduction to Trusted Execution Environments – Global Platform [Электронный ресурс]. – Режим доступа: https://globalplatform.org/wp-content/uploads/2018/05/Introduction-to-Trusted-Execution-Environment-15May2018.pdf, свободный – (09.10.2023)

2. Intel | Data Center Solutions, IoT, and PC Innovation [Электронный ресурс]. – Режим доступа: https://www.intel.com/, свободный – (10.11.2023)

3. Building the Future of Computing – Arm® [Электронный ресурс]. – Режим доступа: https://www.arm.com, свободный – (09.10.2023)

4. The Security Paradox of Complex Systems, 2002. David Woods, Nancy Leveson, Brian Rebentisch. c. 1 - 15.

5. Comparison of Prominent Trusted Execution Environments, 2022. Xiaoyu Zhang [Электронный ресурс]. – Режим доступа: https://elib.uni-stuttgart.de/bitstream/11682/12171/1/Zhang_Xiaoyu_Prominent_TEE.pdf – (22.10.2023)

6. TrustedFirmware-A (TF-A) | ARM [Электронный ресурс]. – Режим доступа: https://www.trustedfirmware.org/projects/tf-a/ – (22.10.2023)

7. Secure Monitor Calling Convention (TF-A) | ARM [Электронный ресурс]. – Режим доступа: https://developer.arm.com/Architectures/SMCCC – (22.10.2023)

8. OP-TEE | ARM [Электронный ресурс]. – Режим доступа: https://www.trustedfirmware.org/projects/op-tee/ – (22.10.2023)

9. Global Platform – TEE Client API Specification [Электронный ресурс]. – Режим доступа: https://globalplatform.org/wp-content/uploads/2010/07/TEE_Client_API_Specification-V1.0.pdf – (22.10.2023)

10. Global Platform – TEE Internal Core API Specification [Электронный ресурс]. – Режим доступа: https://globalplatform.org/specs-library/tee-internal-core-api-specification/ – (22.10.2023)

11. Diffie-Hellman key agreement | IBM [Электронный ресурс]. – Режим доступа: https://www.ibm.com/docs/en/zos/2.1.0?topic=ssl-diffie-hellman-key-agreement – (27.10.2023)

12. FIPS 180-2, Secure Hash Standard [Электронный ресурс]. – Режим доступа: https://csrc.nist.gov/files/pubs/fips/180-2/final/docs/fips180-2.pdf – (27.10.2023)

13. Attestation Services for Intel® Software Guard Extensions [Электронный ресурс]. – Режим доступа: https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/attestation-services.html – (27.10.2023)

14. Keystone Enclave Documentation [Электронный ресурс]. – Режим доступа: https://buildmedia.readthedocs.org/media/pdf/keystone-enclave/docswork/keystone-enclave.pdf – (01.11.2023)

15. Keystone Basics | Keystone Enclave [Электронный ресурс]. – Режим доступа: http://docs.keystone-enclave.org/en/latest/Getting-Started/How-Keystone-Works/Keystone-Basics.html#overview – (01.11.2023)

16. TS-perf: Performance Measurement of Trusted Execution Environment and Rich Execution Environment on Different CPUs, 2021. Kuniyasu Suzaki Kenta Nakajima Tsukasa Oi Akira Tsukamoto

17. Power Analysis Attack – Revealing the Secrets of Smart Cards. Stefan Mangard, Elisabeth Oswald, Thomas Popp, 2007.

18. HARDWARE ATTACK DETECTION AND PREVENTION FOR CHIP SECURITY | Jaya Doef, University of New

Hampshire. [Электронный ресурс] – Режим доступа: https://scholars.unh.edu/cgi/viewcontent.cgi?article=2027&context=thesis – (01.11.2023)

19. Exploiting the DRAM rowhammer bug to gain kernel privileges [Электронный ресурс] – Режим доступа: https://www.blackhat.com/docs/us-15/materials/us-15-Seaborn-Exploiting-The-DRAM-Rowhammer-Bug-To-Gain-Kernel-Privileges.pdf – (01.11.2023)

20. Linux – Open Source Operating System [Электронный ресурс] – Режим доступа: https://www.linux.org/ – (18.02.2024)

21. KVM – Kernel-based Virtual Machine [Электронный ресурс] – Режим доступа: https://linux-kvm.org/ – (18.02.2024)

22. Learn the architecture - AArch64 virtualization [Электронный ресурс] – Режим доступа: https://developer.arm.com/documentation/Trapping-and-emulation-of-instructions – (01.04.2024)

23. vTZ: Virtualizing ARM TrustZone [Электронный ресурс] – Режим доступа: https://ipads.se.sjtu.edu.cn/_media/publications/vtz-security17.pdf (08.04.2024)

24. Arm CoreSight Performance Monitoring Unit Architecture [Электронный ресурс] – Режим доступа: https://developer.arm.com/documentation/ihi0091/latest/ (11.05.2024)

25. Raspberry Pi 4 Model B | Raspberry [Электронный ресурс] – Режим доступа: https://www.raspberrypi.com/products/raspberry-pi-4-model-b (11.05.2024)

26. Raspberry Pi 2 Model B / Raspberry [Электронный ресурс] – Режим доступа: https://www.raspberrypi.com/products/raspberry-pi-2-model-b (11.05.2024)

27. Алгоритм SHA-256 | Google [Электронный ресурс] – Режим доступа: https://support.google.com/google-ads/answer/9004655 (11.05.2024)

28. MongoDB: The Developer Data Platform | MongoDB [Электронный ресурс] – Режим доступа: https://www.mongodb.com/ (11.05.2024)

29. Welcome! - The Apache HTTP Server Project [Электронный ресурс] – Режим доступа: https://httpd.apache.org/ (11.05.2024)