МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Политехнический институт Факультет приборостроения, информационных технологий и электроники	Кафедра «Информационная безопасность систем и технологий»
Специальность 10.05.02 –	Информационная безопасность телекоммуникационных систем
Специализация -	Разработка защищенных телекоммуникационных систем

ДИПЛОМНЫЙ ПРОЕКТ

на тему

ПРОГРАММНОЕ СРЕДСТВО ТЕСТИРОВАНИЯ И УПРАВЛЕНИЯ МОДУЛЕМ ДОВЕРЕННОЙ ПЛАТФОРМЫ, ФУНКЦИОНИРУЮЩЕГО В СРЕДСТВАХ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ, ПРЕДНАЗНАЧЕННЫХ ДЛЯ ОБРАБОТКИ ИНФОРМАЦИИ ОГРАНИЧЕННОГО ДОСТУПА

Студент ___ Аксенов Алексей Александрович

Руководитель инженер АО «ПНИЭИ» _____ М.С.Клейменова

Нормоконтролёр _____ А.Г. Фатеев

Рецензент начальник НИС АО «ПНИЭИ» __ А.Н.Зорин

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой _____ С.Л. Зефиров

Работа защищена с отметкой _____(протокол заседания ГЭК от _____ № ____)

Секретарь ГЭК _____ А.Г. Фатеев

Пенза 2018

Оглавление

Введение 4
1 Актуальность применения и обзор технологий доверенных вычислений 5
1.1 Актуальность применения технологий доверенных вычислений 5
1.2 Обзор технологий доверенных вычислений 6
2 Анализ приложений управления модулем Trusted Platform Module 13
2.1 Анализ приложения Trousers TSS 13
2.2 Анализ приложения TPM Manager 15
2.3 Анализ приложения TPM Tools 20
2.4 Анализ приложения Open SSL Engine 22
3 Архитектура и основные возможности модуля Trusted Platform Module 24
3.1 Общая характеристика модуля доверенных вычислений TPM 24
3.2 Задачи и принцип работы модуля доверенных вычислений TPM 24
3.3 Архитектура модуля доверенных вычислений TPM 25
4 Описание среды окружения программного средства 29
4.1 Установка эмулятора модуля Trusted Platform Module 29
4.2 Установка и настройка программного стека доверенных вычислений TrouSerS Trusted Computing Group Software Stack 32
4.3 Установка приложения TrouSerS TPM Tools 33
4.4 Установка компилятора GNU Compiler Collection 34
5 Разработка программного средства тестирования и управления модулем доверенной платформы 36
5.1 Разработка алгоритма работы программного средства 36
5.2 Разработка программного средства 53
5.3 Отладка программного средства 56
6 Тестирование модуля доверенной платформы, с использованием разработанного программного средства 59
7 Вредные психофизические факторы, влияющие на нервную систему специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие 70

Заключение 72
Список используемых источников 73
Приложение А Текст программных модулей тестирования и управления модулем доверенной платформы 75
Приложение Б Протокол проверки выпускной квалификационной работы в системе Антиплагиат 132

Введение

Стремительное развитие и внедрение средств вычислительной техники в производство значительно увеличивает производительность труда, позволяет обрабатывать большие объемы информации за меньшее время. Но с увеличением количества информации, обрабатываемой в электронном виде, все более актуальной становится проблема обеспечения ее безопасности.

Исключить данные недостатки можно с помощью технологии доверенных вычислений, предназначенной для защиты информации ограниченного доступа, обрабатываемой средствами вычислительной техники.

В выпускной квалификационной работе будут рассмотрены следующие вопросы: актуальность применения и обзор технологий доверенных вычислений, таких как, технология Intel vPro, Bitlocker, Next Generation Secure Computing, технология доверенных вычислений стандартов безопасности Trusted Computing Group. Будет произведен анализ приложений управления модулем, таких как, TPM Manager, TPM Tools, Open SSL TPM Engine, TrouSerS TSS. Будет приведено описание модуля доверенных вычислений, включая общую характеристику модуля, задачи и принцип работы модуля, а также архитектуру модуля доверенных вычислений. Будет приведено описание среды окружения программного средства Trusted Platform Module. Будет разработано программное средство тестирования и управления модулем доверенной платформы, алгоритм работы программного средства. Также будет проведена отладка программного средства и тестирование модуля доверенной платформы, с использованием разработанного программного средства. В заключении, будут описаны вредные психофизические факторы, влияющие на нервную систему специалиста по защите информации, а также методы и средства физической культуры, снижающие их воздействие.

1 Актуальность применения и обзор технологий доверенных вычислений

1.1 Актуальность применения технологий доверенных вычислений

Основной составляющей комплексной системы обеспечения информационной безопасности ресурсов является постоянная и эффективная защита данных ресурсов. Комплексная системы защиты информационных ресурсов включает в себя мероприятия от несанкционированного доступа по различным видам каналов утечки информации, до исключения специальных воздействий, например, блокирования доступа.

В настоящее время средствам вычислительной техники отводится важная роль распределения информационных ресурсов, а также эффективного взаимодействия между участниками обмена информации в сети. Поэтому все больше набирает актуальность вопрос реализации защитных мер для получения наивысшего уровня защиты средств вычислительной техники.

Для защиты информации ограниченного доступа, обрабатываемой средствами вычислительной техники, применяются технологии доверенных вычислений, которые включает ряд компонентов аппаратных и программных средств. Элементы доверенных вычислений включают возможности в микропроцессоре, подсистеме ввода - вывода, наборе микросхем, и иных компонентах платформы. Технология доверенных вычислений совместно с доверенной операционной системой и приложениями может помочь защитить конфиденциальность и целостность данных.

Технология доверенных вычислений обеспечивает универсальную и более безопасную вычислительную среду, способную к выполнению широкого круга операционных систем и приложений.

Созданием и развитием технологии доверенных вычислений занимаются следующие ведущие мировые производители аппаратных средств и программного обеспечения:

- компания Intel - производитель процессоров и наборов микросхем для персональных компьютеров, который предлагает комплексное решение по реализации технологии доверенных вычислений - технологию Intel vPro;

- компания Microsoft - производитель программного обеспечения, который предлагает технологию защиты данных на дисках Bitlocker и технология доверенных вычислений "Безопасная вычислительная база нового поколения" (Next Generation Secure Computing Base – NGSCB);

- международная организация Trusted Computing Group (TCG) - определяет открытые стандарты для доверенных вычислений и технологий безопасности, включая стандартные средства электронных вычислительных машин (ЭВМ) и интерфейсы программного обеспечения для различных вычислительных платформ, периферийного оборудования и других устройств. Спецификации, разработанные TCG, позволяют повысить безопасность вычислительной среды, не ставя под угрозу функциональную целостность, секретность или индивидуальные права.

1.2 Обзор технологий доверенных вычислений

1.2.1 Технология Intel vPro.

Intel vPro — это технология удаленного управления и обеспечения безопасности компьютерами. Особенностью данной технологии является то, что она позволяет управлять персональными компьютерами даже в том случае, если они находятся в выключенном состояние, но при этом подключены к электропитанию.

Компьютеры, которые поддерживают Intel vPro предоставляют простые средства дистанционного управления и защиты от несанкционированного доступа (НСД). Поэтому контроль за ними можно осуществлять с помощью консоли.

Ниже приведен ряд возможностей, предоставляемых ПК, которые поддерживают Intel vPro:

- инвентаризация оборудования и программного обеспечения (ПО) — позволяет отслеживать изменения и проводить инвентаризацию аппаратных и программных активов персонального компьютера (ПК);

- дистанционное устранение проблем — позволяет производить диагностику и решать проблемы дистанционно, независимо от состояния питания ПК и ОС;

- надежные средства обеспечения безопасности, поддерживаемые аппаратно, — позволяет защищать системы с помощью двух новых уровней безопасности, которые используют виртуализацию. [1]

Работа технология Intel vPro представлена ниже.

Компьютеры, которые поддерживают Intel vPro, должны отвечать следующим требованиям. Системная плата на ПК должна иметь специальный контроллер, энергонезависимую память и BIOS, который позволяет производить конфигурирование данной технологии.

Данная технология представляет собой сочетание двух других технологий: технологии удаленного мониторинга (Intel AMT) и технологии виртуализации (Intel VT).

Для удаленного управления компьютерами необходимы два компонента технологии Intel AMT:

- канал для взаимодействия с удаленной консолью;

- энергонезависимая память. [1]

Intel TXT - это аппаратная основа для механизмов, которые подтверждают надежность платформы во время загрузки и запуска, что позволяет надежно оценить вычислительную платформу и уровень ее защиты. Intel TXT компактен и его трудно победить или подорвать, что позволяет обеспечить гибкость и расширяемость для проверки целостности компонентов платформы во время загрузки и запуска, включая BIOS, загрузчик операционной системы и гипервизор. [2]

Intel TXT уменьшает общую поверхность атаки как для отдельных систем, так и для вычислительных пулов. Технология обеспечивает подпись, которая

представляет состояние среды запуска интактной системы. Соответствующую подпись во время будущих запусков можно затем сравнить с этим хорошо известным состоянием, чтобы проверить запуск доверенного программного обеспечения, выполнить системное ПО и обеспечить, чтобы облачная инфраструктура как служба (IaaS) не была подделана. Затем политики безопасности, основанные на статусе доверенной платформы или пула, могут быть ограничены (или разрешены) развертыванием или передислокацией виртуальных машин и данных на доверенные платформы с известными профилями безопасности. Вместо того, чтобы полагаться на обнаружение вредоносных программ, Intel TXT укрепляет доверие к известной программной среде и, таким образом, гарантирует, что выполняемое программное обеспечение не будет скомпрометировано. Это повышает безопасность для устранения ключевых механизмов скрытой атаки, используемых для получения доступа к частям центра обработки данных для доступа или компрометации информации. [2]

1.2.2 Технология защиты данных на дисках Bitlocker.

BitLocker представляет собой технологию шифрования всего тома, которая гарантирует, что данные, которые хранятся на компьютере, не будут получены в случае НСД к ПК в то время, когда ОС не подвергается нагрузки. Данная технология разработана для ПК, которые оснащены совместимыми микросхемами модуля доверенной платформы, а также BIOS. [3]

Данная технология может поддерживать несколько режимов. К таким режимам относятся:

- BitLocker с доверенным платформенным модулем;

- BitLocker с USB-устройством;

- BitLocker с доверенным платформенным модулем и персональным идентификационным номером (ПИН);

- BitLocker с доверенным платформенным модулем и USB-устройством.

Модуль доверенной платформы устанавливается на системную плату ПК и использует для взаимодействия с другими компонентами компьютера

аппаратную шину. Компьютеры, которые оснащены данным модулем, могут создавать ключи шифрования и шифровать данные так, что их можно было расшифровать только с помощью модуля доверенной платформы.

Данная технология позволяет шифровать данные путем шифрования всего тома с ОС. Если ключ отсутствует, то нарушитель не сможет получить данные с этого компьютера, путем анализа какой-либо части диска. [3]

Механизмы восстановления доступны полномочным пользователям при возникновении определенных условий.

После того, как была пройдена проверка службой BitLocker доступа к защищенному тому с ОС драйвер фильтра файловой системы BitLocker использует полный ключ шифрования тома. [3]

1.2.3 Технология Next Generation Secure Computing.

База Secure Computing Base следующего поколения (NGSCB) является частью ОС Microsoft, в которой используется модуль доверенной платформы Trusted Platform Module (TPM).

TPM хранит информацию, относящуюся к основной системе, такую как ключи шифрования, цифровые сертификаты и пароль. [4]

Первоначальная версия NGSCB называлась Palladium.

Palladium - это технология, которая включает в себя аппаратные и программные компоненты, позволяющие повысить уровень безопасности данных.

Основным компонентом технологии Palladium является специальный сегмент в процессоре, который содержит в себе уникальный идентификатор процессора и ключ шифрования. Кроме того, аппаратная часть будет включать так называемые доверенное пространство и герметичное хранилище. [5]

Аппаратная часть Palladium является основой для построения доверенной программной среды (Distributed Trusted Software). [5]

Доверенная подсистема управления - это интерфейс для управления функциями Palladium. Подсистема управления Palladium работает в доверенном пространстве на уровне ядра ОС.

1.2.4 Технология доверенных вычислений стандартов безопасности Trusted Computing Group.

Благодаря открытым стандартам и спецификациям Trusted Computing Group обеспечивает безопасные вычисления. Преимущества технологий TCG включают защиту критически важных для бизнеса данных и систем, надежную аутентификацию и надежную защиту пользовательских идентификаторов, а также установление сильной идентификации и целостности сети. Надежное оборудование и приложения сокращают общую стоимость владения и поддерживают соответствие нормативным требованиям. Практически все корпоративные ПК, многие серверы и встроенные системы включают TPM; в то время как сетевое оборудование, приводы и другие устройства и системы используют другие спецификации TCG, включая самошифрующие диски и спецификации сетевой безопасности. [6]

Программное обеспечение и приложения доступны для многих разработчиков программного обеспечения. Спецификация TPM 2.0 была принята в качестве международного стандарта Международной организацией по стандартизации Международной электромеханической комиссией.

Сетевые устройства и службы, поддерживающие надежную сетевую связь и спецификацию контроля доступа, доступны от ряда поставщиков. И, самошифрующиеся диски, основанные на спецификациях TCG, доступны от производителей дисков на жестких дисках, SSD и корпоративных форматах хранения данных со многими доступными опциями ПО для управления.

Архитектура доверенных вычислений стандартов безопасности Trusted Computing Group приведена на рисунке 1.

В основе технологии доверенных вычислений, предлагаемой TCG, лежит концепция построения корневой цепи доверия. Базовым элементом для построения цепи доверия является модуль доверенной платформы TPM. [6]

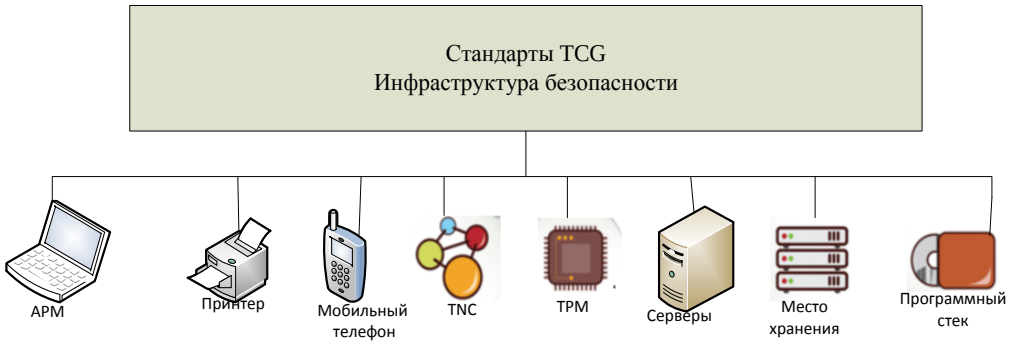


Рисунок 1 – Архитектура доверенных вычислений стандартов безопасности TCG

TPM модуль доверенной платформы — это криптопроцессор, в котором хранятся криптографические ключи для защиты информации (ЗИ), а также общие наименование реализаций данной спецификации. [7]

Основной целью TCPA являлось улучшение механизмов безопасности и доверия в компьютерах. В 2001 году была произведена публикация спецификации TCPA на тот момент версии 1.0, в которой были определены необходимые требования, которые должен выполнять модуль доверенной платформы TPM. Спустя некоторое время спецификация была пересмотрена со стороны практического применения модуля доверенной платформы и летом того же года была выпущена спецификация TPM версии 1.1. После того как была опубликована версия TPM 1.1 в 2003 году была создана организация (TCG), которая заменила TCPA, но при этом продолжила развивать выпущенные спецификации модуля доверенной платформы. В этом же году была опубликована спецификация TPM версия которой, на тот момент стала 1.2. Данная версия уже по существу описывала функциональность TPM. [7]

Модуль доверенной платформы TPM, содержит в себе криптопроцессор, который обеспечивает средства безопасного создания ключей шифрования, которые позволяют ограничить использование ключей, с такой же степенью неповторимости, как и генератора случайных чисел (ГСЧ). Модуль доверенной платформы обладает следующими возможностями, а именно: удалённой аттестацией, привязкой и надежной защитой хранение данных. Криптопроцессор позволяет шифровать данные так, что они могут быть расшифрованы только на

компьютере, где было произведено шифрование. Привязка шифрует данные уникальным ключом подтверждения TPM RSA, который записан в чип при его производстве.

Модуль доверенной платформы TPM используется для подтверждения подлинности аппаратных средств. Так как каждый чип TPM уникален для определенного устройства, то становится возможным однозначное установление и определение подлинности платформы. [7]

Возможности TPM широко используют мировые компании по производству ПО и аппаратных средств для реализации различных технологий ЗИ. Примерами являются технология Intel vPro и технология Bitlocker.

2 Анализ приложений управления модулем Trusted Platform Module

2.1 Анализ приложения Trousers TSS

В данном вопросе рассмотрены некоторые основные компоненты с открытым исходным кодом платформы доверенных вычислений, такие как:

- Trousers TSS;
- TPM Manager;
- TPM Tools;
- Open SSL TPM Engine.

Программный стек Trusted Computing Software Stack (TSS), обеспечивает интерфейс API операционным системам и приложениям так, чтобы они могли использовать функциональные возможности, которые обеспечивает TPM. Это позволяет разным приложениям и операционным системам, работать как виртуальные машины, чтобы взаимодействовать друг с другом, пока они соответствуют спецификации TSS.

Основная цель TSS состоит в том, чтобы мультиплексировать доступ к TPM, так как TPM ограничивает ресурсы и может только связываться с одним клиентом. Кроме того, не все функции, связанные с доверительными вычислениями, требуют доступа TPM. Такие функции находятся в спецификации TSS.

TrouSerS - реализация TSS с открытым исходным кодом. Последняя выпущенная версия стабильна и реализует версию 1.1b взаимодействия через интерфейс версии 1.2 TSS TPM.

Архитектура программного стека Trousers TSS представлена на рисунке 2.

TrouSerS обладает следующими уровнями ПО TCG:

- TSS Service Provider (TSP) – служба платформы провайдера стека;
- TPM Device Driver Library (TDDL) – библиотека драйвера TPM;
- TSS Core Services (TCS) – служба ядра TSS.

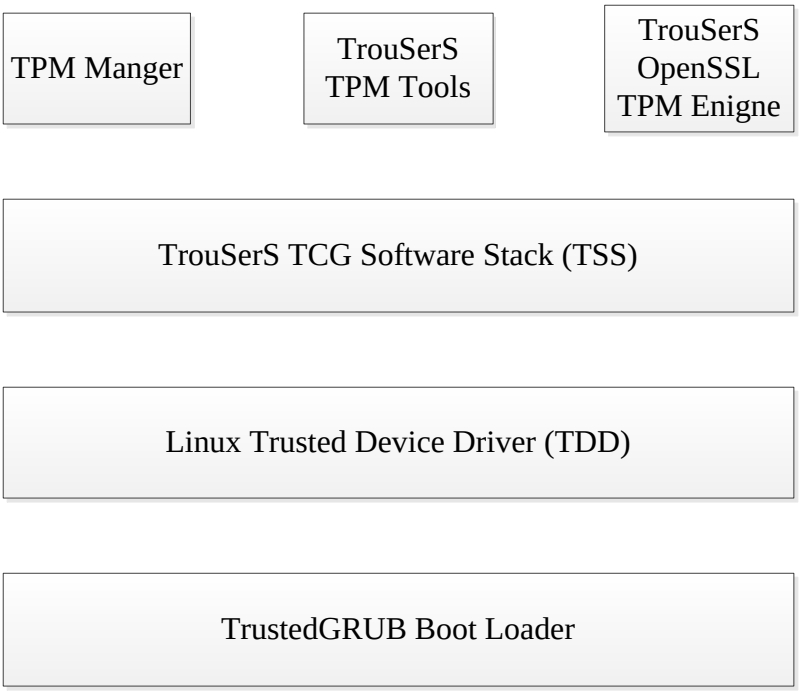


Рисунок 2 – Архитектура программного стека

Взаимодействие данных уровней приведено на рисунке 3.

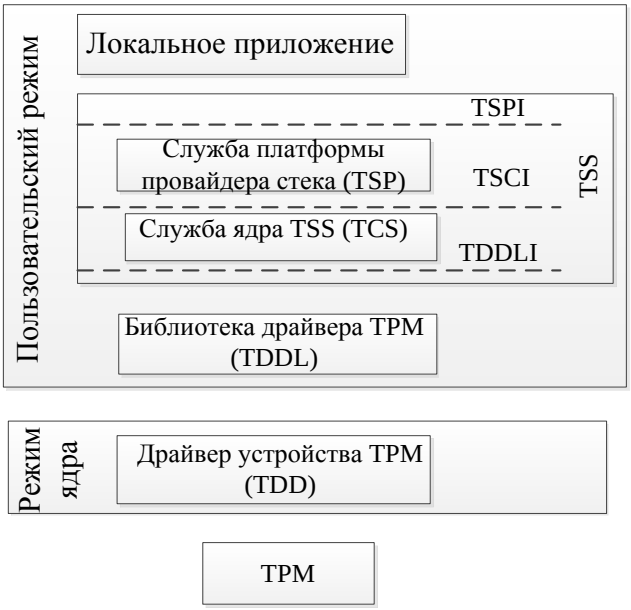


Рисунок 3 – Взаимодействие уровней ПО TCG

TSP и TSPI.

Каждое приложение, использующее функциональность TPM, должно использовать TSP через свой интерфейс, TSS Service Provider Interface (TSPI). Каждое приложение использует свой собственный TSP, который загружен как библиотека в любой процесс приложения. TSP обеспечивает высокоуровневые

функции TCG, а также вспомогательные службы, такие как проверка подписи или хеширование.

TSP - необходим для каждой реализации TSS. TSP может быть дискретным модулем; возможно интегрировать TSP в другие модули платформы. Один аспект TSP – TSP Context Manager (TSPCM), который обеспечивает динамические дескрипторы, позволяющие эффективно использовать ресурсы приложения и TSP.

Второй аспект TSP Cryptographic Functions (TSPCF), который обеспечивает, чтобы полностью использовались защищенные функции в TPM.

TCS и TCSI.

TCS обеспечивает единый набор операций ко всей работе TSP платформы. Основная задача TCS состоит в том, чтобы мультиплексировать доступ TPM. Кроме того, TCS обеспечивает операции, чтобы сохранить, управлять и защитить ключи.

TDDL и TDDLI.

TDDL, к которой получают доступ через TCG Device Driver Library Interface (TDDLI), обеспечивает уникальный интерфейс для различных реализаций драйвера TPM. Кроме того, это обеспечивает переход между привилегированным режимом и пользователем.

TDD и TDDI.

Драйвер устройства TPM включает код, который понимает определенное поведение TPM. Это - единственный модуль, который связывается непосредственно с TPM. Он получает потоки байтов от TDDL и пересылает их к TPM, возвращая его ответ назад на TDDL. Несмотря на то, что спецификация TSS упоминает интерфейс TDD TDDi, текущая спецификация TSS не определяет его.

2.2 Анализ приложения TPM Manager

Целью TPM Manager является реализация программного обеспечения управления TPM с открытым исходным кодом, который обеспечивает простой в

использовании графический интерфейс пользователя. В настоящее время TPM Manager может использоваться в соответствии с Linux, в то время как более поздние версии должны также быть применимыми как изолированное приложение.

Так как TPM Manager основывается на некоторых виджетах KDE, которым соответствуют заголовочные файлы в библиотеках KDE, и QT, то необходимо прописывать пути к необходимым библиотекам.

У TPM Manager есть два уровня:

- уровень Graphical User Interface (GUI);

- уровень MicroTSS.

В то время как уровень GUI обеспечивает компоненты пользовательского интерфейса, на основе виджетов KDE19 и платформ QT20, уровень MicroTSS предлагает простой, объектно-ориентированный интерфейс функциям базового TPM.

Несмотря на то, что текущая реализация MicroTSS основывается на TrouSerS TSS, абстракция, обеспеченная уровнем MicroTSS, допускает поддержку со стороны альтернативных реализаций TSS, предлагаемых третьей стороной или от требуемых функций. Последнее условие особенно важно, если менеджер TPM используется, как часть ядра безопасности.

TPM Manager обеспечивает выполнение функций, приведенных на рисунке 4 и таблице 1 [14].

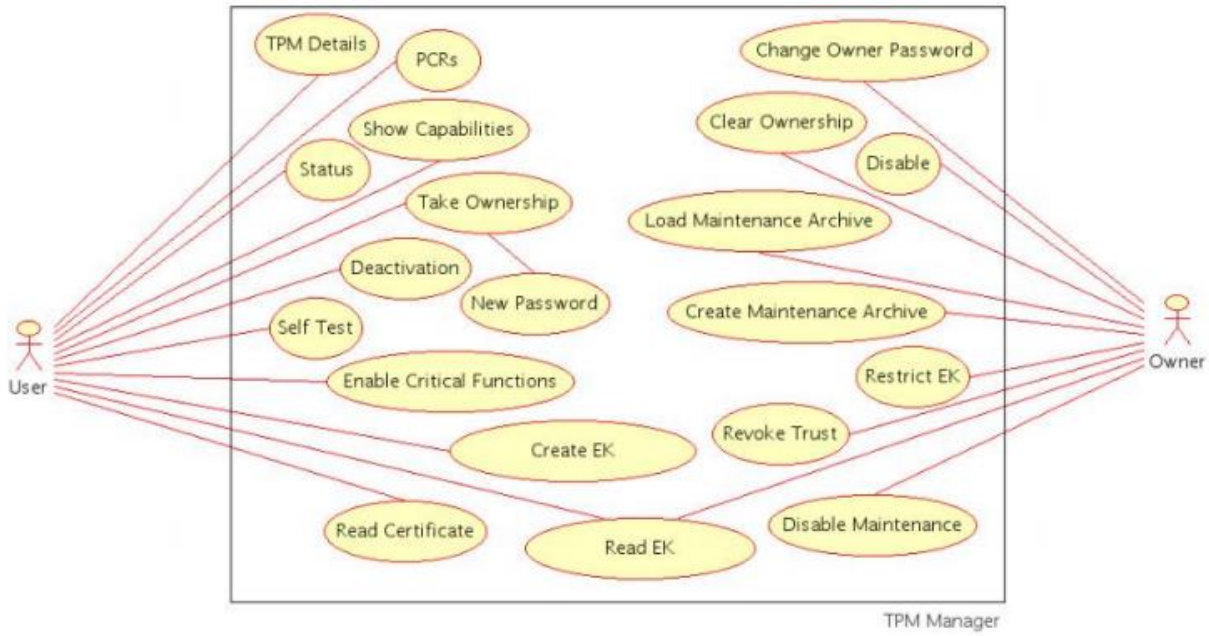


Рисунок 4 – Функции TPM Manager

Таблица 1 – Функции TPM Manager

Команды	Описание	Исполнитель команды пользователь (User)/ владелец (Owner)
TPM Details	Отображение подробной информацию о TPM (производитель, версия, встроенное ПО) и TSS (производитель, версия)	User
Status	Отображение состояния TPM (включение TPM, активация, установка владельца, доступность ключа подтверждения EK, поиск драйвера TPM, поиск TSS)	User

Продолжение таблицы 1

Команды	Описание	Исполнитель команды пользователь (User)/ владелец (Owner)
Show Capabilities	Отображение возможностей TPM (количество PCR, доступные свойства)	User
Take Ownership	Конфигурирование владельца TPM и установка встроенной функции безопасности	User
Deactivation	Переключение режима работы TPM в активированный или временно деактивированный режим	User
New Password	Создание нового пароля пользователем или владельцем	User
Self Test	Выполнение полного самотестирования всех внутренних функций TPM	User
Enable Critical Functions	Включение критических функций	User

Продолжение таблицы 1

Команды	Описание	Исполнитель команды пользователь (User)/ владелец (Owner)
Create EK	Создание ключа подтверждения (EK)	User
Read EK	Чтение ключа подтверждения (EK)	User, Owner
Read Certificate	Чтение сертификата	User
Change Owner Password	Изменение пароля, используемого для аутентификации владельца TPM	Owner
Disable	Переключение режима работы TPM во включенный или отключенный режим	Owner
Clear Ownership	Удаление владельца TPM и установка заводских настроек в TPM	Owner
Load Maintenance Archive	Загрузка резервного копирования ключа SRK	Owner
Create Maintenance Archive	Создание резервного копирования ключа SRK	Owner

Продолжение таблицы 1

Команды	Описание	Исполнитель команды пользователь (User)/ владелец (Owner)
Restrict EK	Отключение чтения ключа EK без авторизации владельца	Owner
Revoke Trust	Удаление ключа EK	Owner
Disable Maintenance	Отключение обслуживания характеристик	Owner

2.3 Анализ приложения TPM Tools

Проект TrouSerS TPM Tools включает в себя инструментальные средства по работе с TPM.

TPM Tools поддерживает команды, разрешающие администратору платформы управлять и диагностировать объект TPM. В дополнение к этому пакет содержит команды для использования возможностей, доступных в интерфейсе TPM PKCS#11, реализованном в openCryptoki проекте, реализации PKCS#11 для Linux.

Некоторые компоненты, которые включает в себя это приложение представлены ниже.

К TSS получают доступ через интерфейс TSP, реализованный библиотекой libtspi.so. Несмотря на то, что эта библиотека реализована в ANSI C, это основывается на объектно-ориентированном проекте. Поэтому весь TSPI функционирует с одной или несколькими ссылками, адресуя определенные экземпляры класса.

Context - класс, который представляет соединение с TCS, работающим или на локальной, или на удаленной вычислительной платформе TCG. Методы,

обеспеченные в этом классе, управляют ресурсами, памятью и другими объектами. Другой фокус должен обеспечить соединения с TCS и персистентной базой данных хранения. Взаимодействие между TCS и TSP в соответствии с Linux понято сокетами TCP, где TCS находится на номере порта 30003. Большинство функций определено в классе TPM, который представляет владельца TPM, который обеспечивает основное управление и сообщающие функции.

Policy класс может использоваться, чтобы сконфигурировать стратегические параметры настройки.

Key класс предлагает обработку ключа TCG и функциональность.

В текущей версии Инструменты TPM поддерживают семнадцать инструментов командной строки, которые приведены в таблице 2.

Таблица 2 – Основные команды TPM Tools

Команды	Описание
tpm_changeownerauth	Изменение данных авторизации, связанных с владельцем или SRK
tpm_clear	Возврат TPM в состояние по умолчанию (выключен, деактивирован, нет владельца)
tpm_createek	Создание пары ключей подтверждения в TPM
tpm_getpubek	Отобразить открытую часть ключа подтверждения TPM
tpm_resetdalock	Сброс блокировки атак по словарю для пользователя (требуется аутентификация владельца)
tpm_restrictpubek	Ограничить способность отобразить открытую часть ключа подтверждения владельцу
tpm_revokeek	Отменить пару ключей подтверждения TPM

Продолжение таблицы 2

Команды	Описание
tpm_sealdata	Опечатать входные данные системы
tpm_selftest	Запрос TPM на выполнение самотестирования и отчета о результатах
tpm_setactive	Изменение состояния активного TPM
tpm_setclearable	Выключение операций очистки TPM
tpm_setenable	Изменение состояния включенного TPM
tpm_setoperatorauth	Установка данных авторизации оператора в TPM
tpm_setownable	Изменение разрешения установки владельца TPM
tpm_setpresence	Изменение состояний и установок режима физического присутствия в TPM
tpm_takeownership	Установка владельца TPM
tpm_version	Получение данных о версии и производителе TPM

Команды TPM Tools обеспечивают выполнение следующих функций:

- управление владельцем TPM;
- управление ключом подтверждения;
- управление конфигурацией.

2.4 Анализ приложения Open SSL Engine

OpenSSL - инструментарий с открытым исходным кодом, который реализует уровень защищенных сокетов (SSL v2/v3) и протоколы безопасности транспортного уровня (TLS v1), которые включают в себя необходимые криптографические операции. Инструментарий основывается на библиотеке SSLeay, которую разработали Eric A. Young и Tim J. Hudson в соответствии с

лицензией стиля Apache. И поэтому возможно использование как для коммерческих, так и для некоммерческих целей.

Библиотека OpenSSL поддерживает создание, управление и использование криптографических модулей как контейнеры для реализаций криптографических алгоритмов. Так как эти объекты могут быть загружены динамично, OpenSSL может быть расширен альтернативными реализациями программного обеспечения или аппаратными модулями, такими как смарт-карты или TPM.

Механизм TPM соединяет TPM с OpenSSL через TSS. Для более корректной связи с TPM, сам механизм OpenSSL TPM пользуется совместно с библиотекой libtspi.so, которая является частью TSS.

OpenSSL TPM имеет следующие функциональные возможности:

- альтернативный алгоритм RSA, (EC) DSA и (EC) DH;
- симметричные алгоритмы шифрования;
- алгоритмы хеширования;
- загрузка ключа;
- хранение криптографических ключей и сертификатов.

3 Архитектура и основные возможности модуля Trusted Platform Module

3.1 Общая характеристика модуля доверенных вычислений TPM

TPM – это микроконтроллер, который реализует работу функций по обеспечнию безопасности ПК, при этом использует ключ шифрования. Установка чипа TPM происходит на материнской плате ПК. Данный чип осуществляет взаимодействие с другими компонентами компьютера через системную шину. [8]

Хэширование для системы происходит с помощью алгоритма SHA1 (Secure Hash Algorithm). Значение хеша складывается из информации, которая получена от всех компонентов компьютера, в сочетании с программными элементами данного ПК. [8]

Запуск компьютера возможен в том случае, если TPM получает правильное значение хеша, такое состояние называется проверенным состоянием компьютера. В этом состояние ОС получает доступ к корневому ключу шифрования. Данный ключ необходим для того, чтобы выполнялась работа приложений, а также был доступ к данным, которые защищает TPM. [8]

3.2 Задачи и принцип работы модуля доверенных вычислений TPM

Основная задача TPM состоит в создании такого надежного компьютера, в котором осуществляется проверка и защита всех процессов связи, аппаратного и программного обеспечение. Защита связи заключается во взаимодействии между отдельными частями системы (например, ОС). [7]

Также функцией TPM является проверка целостности и подлинности данных. Только авторизированные пользователи могут иметь доступ к данным, при этом также должна быть обеспечена безопасная передача информации. Это позволяет защитить систему от вредоносных программ, которые позволяют изменять данные, без ведома пользователя.

TMP можно также использовать для обеспечения безопасности мобильных телефонов, карманных персональных компьютеров, устройств ввода и дисковых

накопителей. Вместе с ним можно применять устройства биометрической идентификации.

3.3 Архитектура модуля доверенных вычислений TPM

Чип TPM должен удовлетворять требованияем минимального набора алгоритмов и протоколов, которые описаны в документации TCG. Для реализации дополнительных алгоритмов и протоколов необходимо, чтобы производители вносили их описание в соответствующую документацию.

В архитектуре чипа реализованы следующие защитные механизмы:

- защищённое управление памятью;
- шифрование шины и данных;
- тестирование режимов блокирования;
- активное экранирование.

Архитектура модуля доверенных вычислений TPM приведена на рисунке 5. [7]

Ниже приведено описание компонентов данной схемы.

Криптографический процессор выполняет операции внутри TPM, такие как:

- генерация асимметричных ключей (RSA);
- асимметричное шифрование/расшифрование (RSA);
- хеширование (SHA-1);
- генерация случайных чисел.

TPM использует эти возможности для генерации случайных последовательностей, генерации асимметричных ключей, цифровой подписи и конфиденциальности хранимых данных. Также TPM поддерживает симметричное шифрование для внутренних нужд. Все хранимые ключи по силе должны соответствовать ключу RSA длиной 2048 бит. [7]

Блок SHA-1 используется для вычисления подписей, создания блоков ключей и других целей общего назначения. Хеш-интерфейсы доступны вне

доверенного платформенного модуля, что позволяет окружению иметь доступ к хэш-функции.

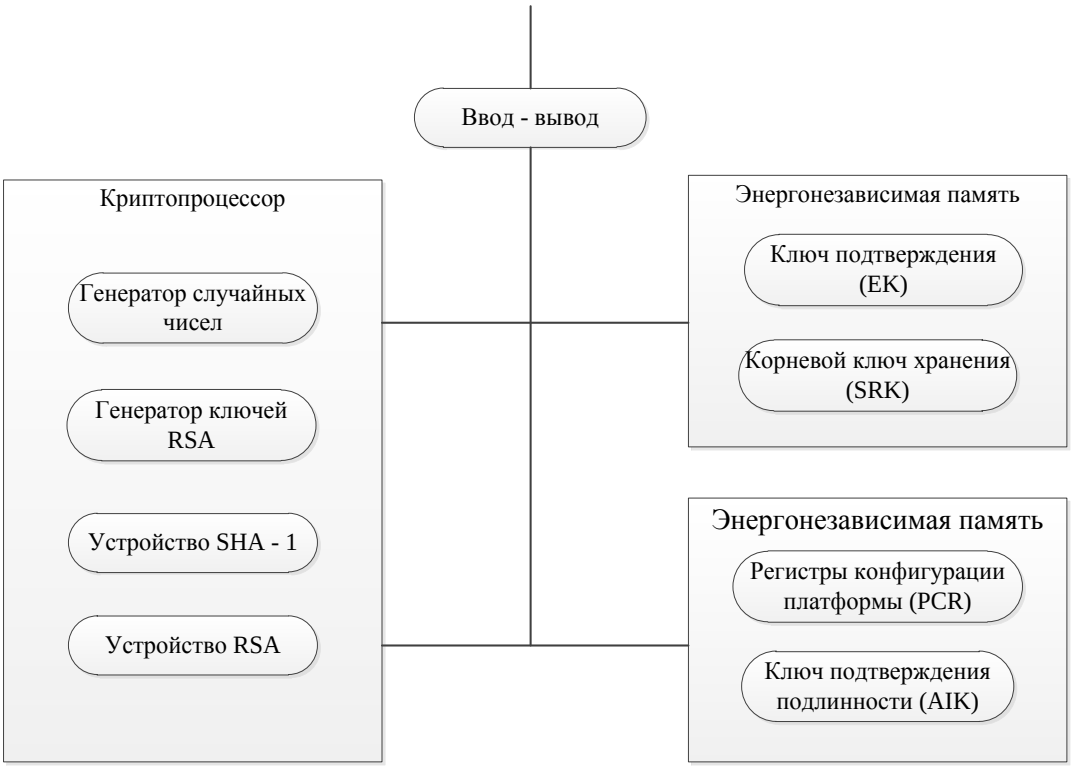


Рисунок 5 – Архитектура модуля доверенных вычислений TPM

Генератор ключей RSA необходим для создания пар ключей для данного алгоритма. В спецификации не указано требований для времени генерации ключей, в ней указывается только минимальная длина ключа, которая необходима для шифрования. Длина ключа должна составлять 2048 бит. [9]

Устройство RSA используется для генерации цифровых подписей и шифрования. Для реализации данного алгоритма в спецификации не указано никаких ограничений. Поэтому производители при разработке модуля доверенной платформы могут ипользовать любой им удобный метод.

Random Number Generator (RNG) – генератор случайных чисел, используется для генерации ключей и случайностей в сигнатурах подписях. Доверенный платформенный модуль должен быть способным предоставить 32 случайных бита на каждый вызов генератора. [9]

RNG крипточип состоит из следующих компонентов: источник энтропии, коллектор энтропии, регистр состояния и смешивающая функция.

Источник энтропии – это процесс или процессы, вырабатывающие статистически независимые сообщения.

Коллектор энтропии — процесс, который собирает эту информацию, удаляет смещение и выравнивает выходные данные. Энтропия должна передаваться только регистру состояния. Реализация регистра состояния может использовать два регистра: энергозависимый и независимый. [9]

Энергонезависимая память используется для хранения ключа подтверждения, корневого ключа – Storage Root Key (SRK), авторизационных данных. В спецификации объем данного типа памяти ограничен 1280 байтами.

Endorsement Key (EK) – ключ подтверждения, это ключ RSA размером 2048 бит, идентифицирующий чип, а также все устройство. Он является основным компонентом модуля доверенной платформы TPM. Открытая часть ключа PUBEK, закрытая — PRIVEK.

PRIVEK не должен быть доступен вне чипа, а также не использоваться для генерирования подписей. PUBEK хранится в сертификате, используется только для установления владельца TPM.

Attestation Identity Keys (AIK) – ключ подтверждения подлинности, это ключ RSA длиной 2048 бит, используемый только для подписей. Использование его для шифрования запрещено спецификацией. Использование AIK от одного модуля на другом запрещено.

Ключевая иерархия TPM приведена на рисунке 6.

Platform Configuration Registers (PCR) – регистры конфигурации платформы, это уникальные записи доверенного платформенного модуля, в которых в зашифрованном виде содержится вся информация о состоянии целостности системы.

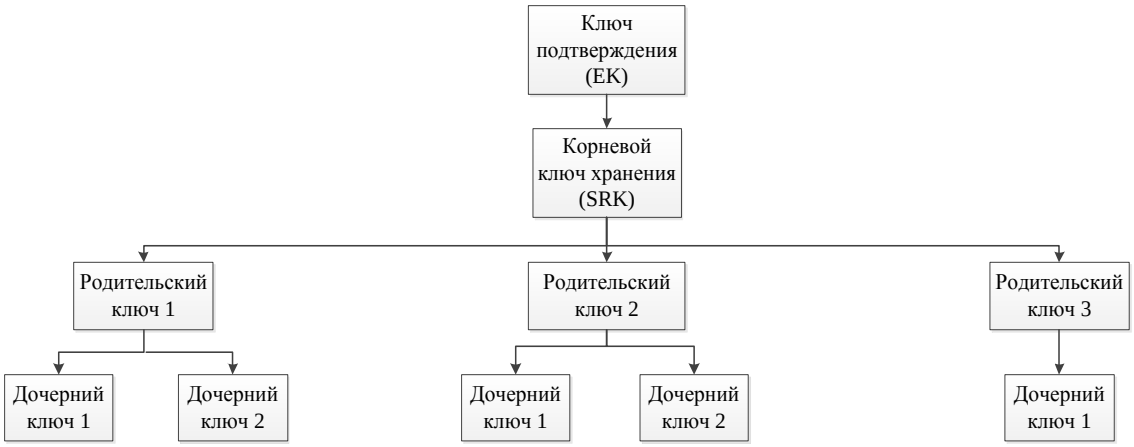


Рисунок 6 – Ключевая иерархия TPM

Регистры конфигурации платформы могут храниться как в энергонезависимой, так и в энергозависимой памяти. Регистры сбрасываются при запуске или при перезагрузке ОС.

Всего в спецификации указано 16 регисторов, которые содержат по 20 бит информации. Регистры с 0 по 7 зарезервированы для требований TPM. Регистры с 8 по 15 доступны для использования ОС. [10]

4 Описание среды окружения программного средства

4.1 Установка эмулятора модуля Trusted Platform Module

Установка эмулятора TPM проводилась в ОС Centos 7.0.

Для этого был установлен дистрибутив на виртуальную машину и далее были проведены действия по настройке ОС, а именно обновление приложений до последних версий.

Рабочий стол ОС Centos 7.0 представлен на рисунке 7.

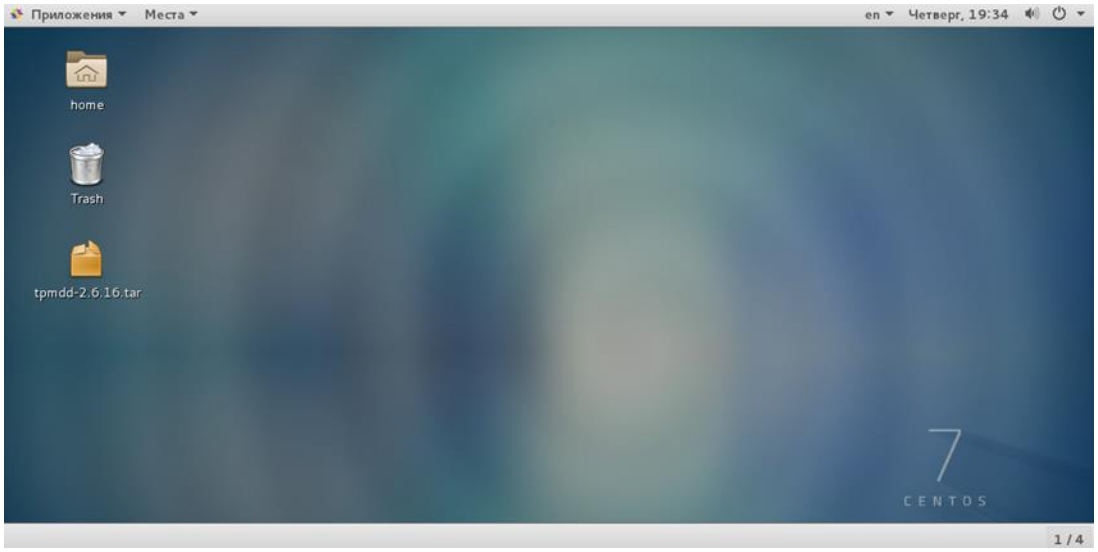


Рисунок 7 – Рабочий стол ОС Centos 7.0

Для корректной работы эмулятора TPM необходимо установить и настроить такие программы, как «cmake» и «gmp»

CMake (от англ. cross platform make) — это кроссплатформенная система автоматизации сборки программного обеспечения из исходного кода.

Текст вводимых команд для установки cmake:

```
#./bootstrap
#make
#make install
```

Проверка установки приложения приведено на рисунке 8.

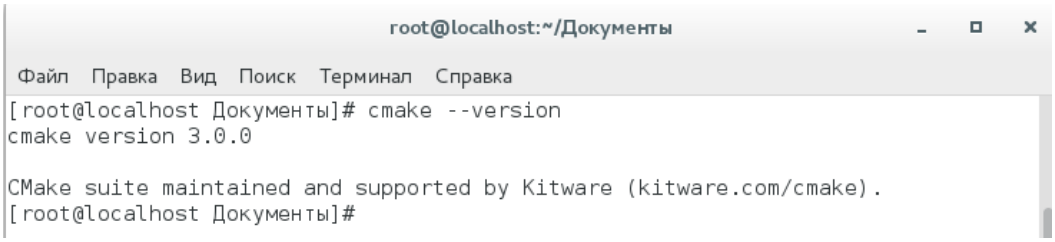


Рисунок 8 – Проверка установки приложения cmake

На данном рисунке показана версия программы cmake.

Далее было установлено приложение gmp.

GMP - это библиотека, которая дает возможность работать с целыми, рациональными числами и числами с плавающей точкой.

Для этого были введены следующие команды:

```
#./configure
#make
#make check
#make install
```

Для установки приложения TPM Emulator версии 0.7.4 были введены следующие команды:

```
# tar -xvzf tpm_emulator-0.7.4.tar.gz
# cd tpm_emulator-0.7.4
# mkdir build
# cd build
# cmake ../
# make
# make install
```

Для настройки эмулятора TPM необходимо ввести следующие команды:

```
# tpmd deactivated
# killall tpmd
# tpmd clear
# depmod –a
```

Результат настройки представлен на рисунке 9.

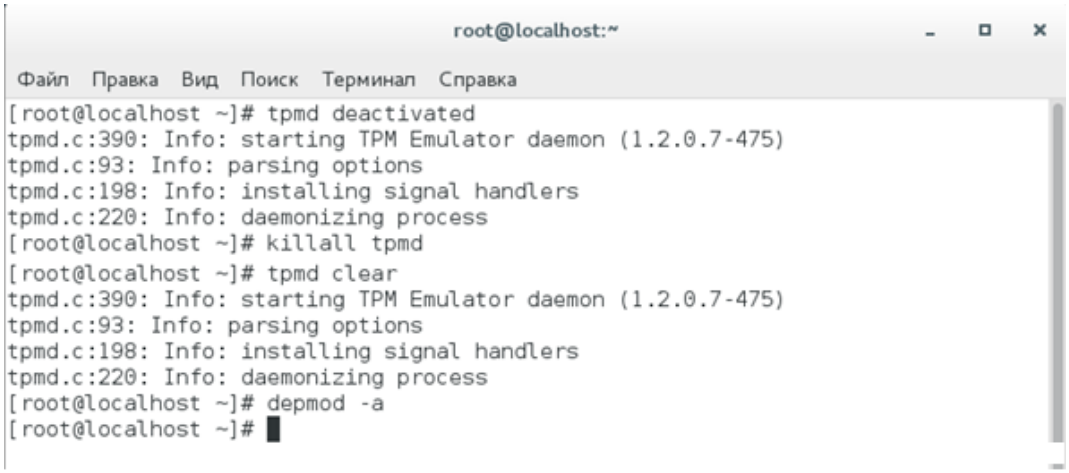


Рисунок 9 – Настройка эмулятора TPM

Для того чтобы запустить эмулятор TPM необходимо ввести следующие команды:

```
#rm /var/run/tpm/tpmd_socket:0
#modprobe tpmd_dev
#tpmd –f –d
```

Параметр - f означает отображение эмулятора TPM в активном окне.

Параметр - d - включение режима отладки.

Результат работы программы эмулятора TPM приведен на рисунке 10.

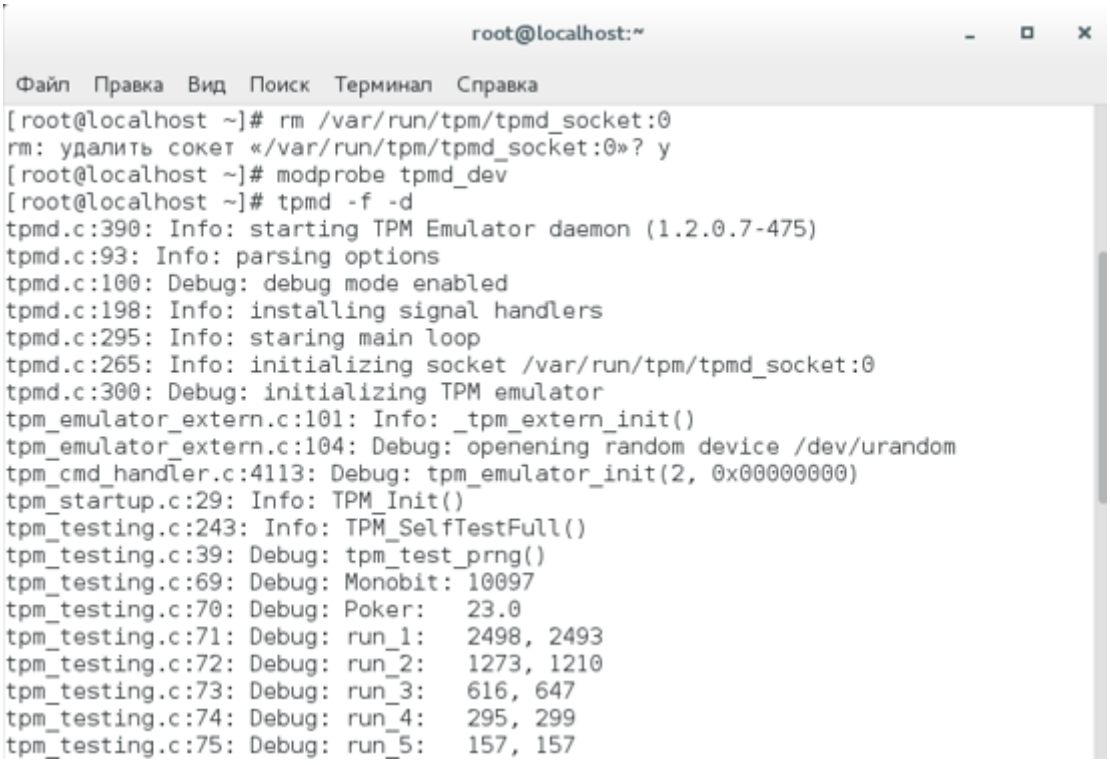


Рисунок 10 – Работа эмулятора TPM

Таким образом, после установки необходимых программ, TPM эмулятор работает корректно.

4.2 Установка и настройка программного стека доверенных вычислений TrouSerS Trusted Computing Group Software Stack

Взаимодействие TPM с программным обеспечением ПЭВМ производится через программный стек доверенных вычислений TSS, называемый по-другому сервисом (службой) поддержки доверенной платформы.

Для корректной работы программного стека с эмулятором TPM в ОС Linux необходимо установить следующие приложения:

- openssl – devel;
- gtk – devel;
- libtool.

Данные приложения были установлены с помощью команды «yum».

Для установки программного стека TrouSerS TSS необходимо ввести следующие команды:

```
# sh bootstrap.sh
# export PKG_CONFIG_PATH=/usr/lib64/pkgconfig
# CFLAGS="-L/usr/lib64-L/opt/gnome/lib64"LDFLAGS="-L/usr/lib64
\-L/opt/gnome/lib64"./configure --libdir="/usr/local/lib64"
# make
# make install
```

Для того, чтобы запустить программный стек необходимо ввести следующую команду:

```
#tcsd –e –f
```

Параметр –e - запуск программного стека для взаимодействия с TPM через программный интерфейс сокетов TCP/IP.

Параметр –f - отображение программного стека в активном окне.

Результат работы программного стека и эмулятора TPM приведен на рисунке 11.

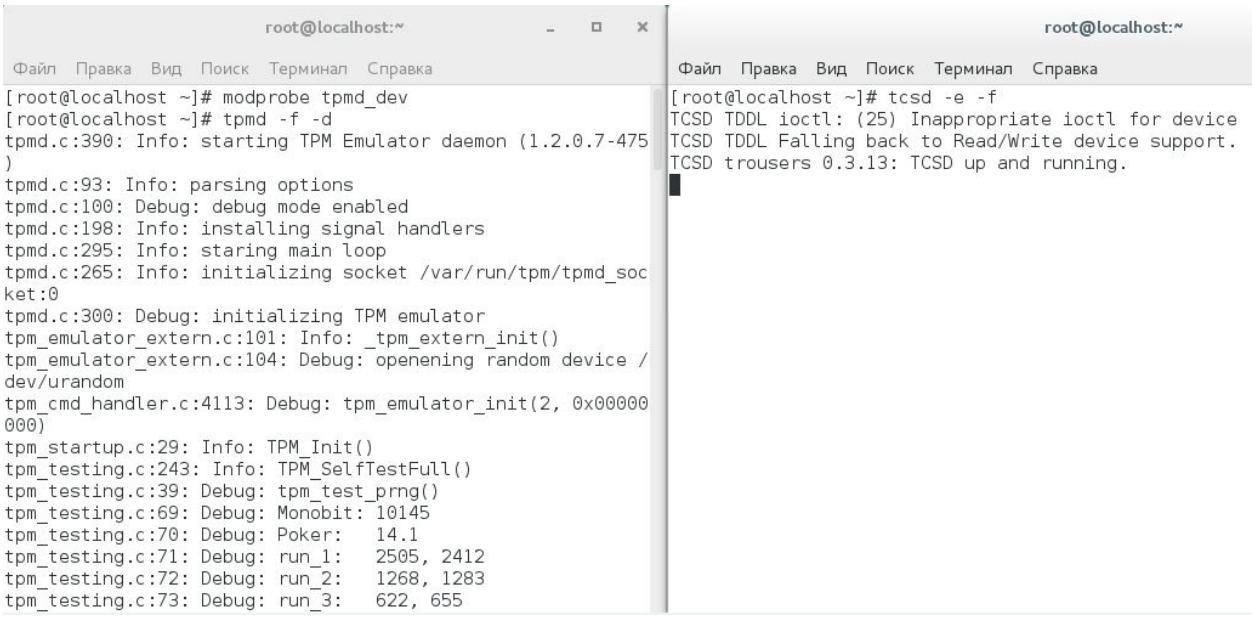


Рисунок 11 – Результат работы программного стека и эмулятора TPM

4.3 Установка приложения TrouSerS TPM Tools

Также основным компонентом работы с программным стеком является приложение TPM Tools.

Для установки данного приложения была использована команда «`yum`».

После того, как установилось данное приложение необходимо ввести следующие команды:

```
# tpm_version
# tpm_takeownership -z
```

Результат работы приложения TPM Tools представлен на рисунке 12.

Команда `tpm_version` дает информацию об установленной версии TPM.

Команда `tpm_takeownership` устанавливает пароль владельца TPM и параметр –z устанавливает пароль корневого ключа хранения SRK по умолчанию (двадцать нулей).

Команда `tpm_getpubek` - получение открытой части ключа подтверждения EK.

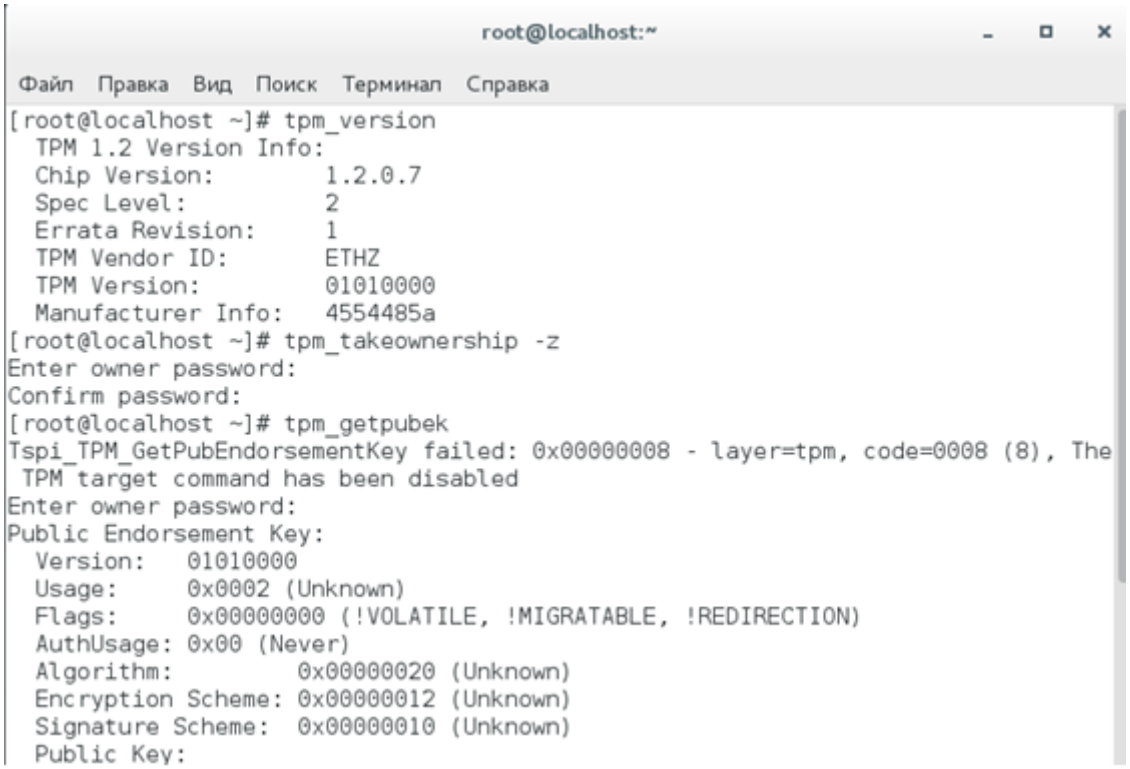


Рисунок 12 – Результат работы TPM Tools

Таким образом настройка и установка программного стека выполнена в полном объеме.

4.4 Установка компилятора GNU Compiler Collection

GNU Compiler Collection (обычно используется сокращение GCC) — набор компиляторов для различных языков программирования, разработанный в рамках проекта.

Для проверки правильности установки «gcc», была создана программа, которая проверяет корректность работы.

Результат установки «gcc» приведен на рисунке 13.

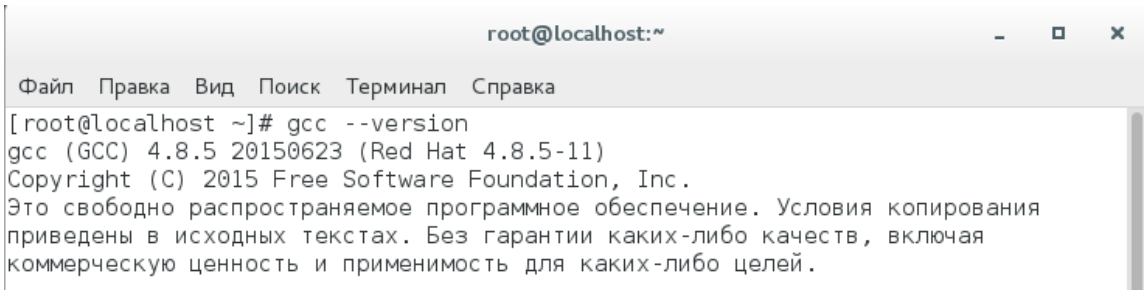


Рисунок 13 – Версия gcc

На данном рисунке показана версия компилятора gcc.

Для проверки работоспособности было выполнено следующее.

Был создан отдельный каталог под названием «hello». В нем был создан текстовый файл с названием «hello.c». В данном текстовом файле была написана программа. В каталоге проекта была открыта консоль и введена следующая команда `gcc hello.c`.

После этого в каталоге появляется новый файл с расширением `.out`.

Файл с таким расширением является исполняемым файлом. Далее был запущен исполняемый файл командой `./a.out`.

Результатом программы является текст `Hello world!`

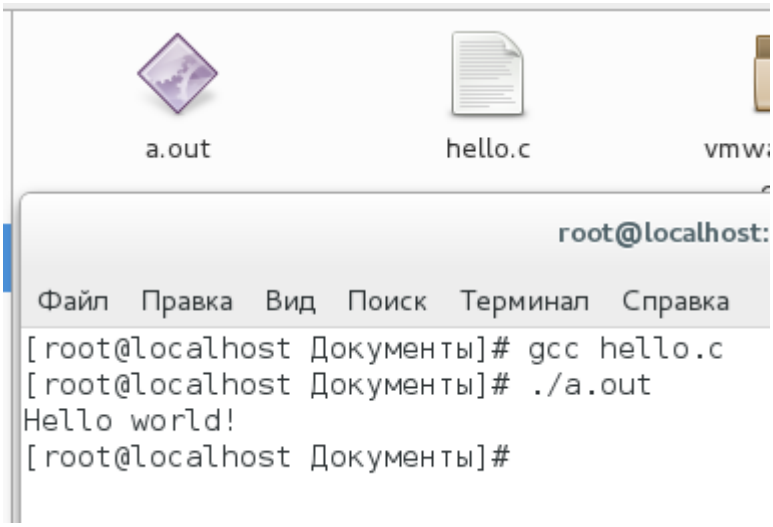


Рисунок 14 – Работоспособность gcc

5 Разработка программного средства тестирования и управления модулем доверенной платформы

5.1 Разработка алгоритма работы программного средства

5.1.1 Алгоритм модуля хеширования данных.

Данный модуль позволяет проверять правильность шифрования данных с ключом, а также записывать зашифрованные данные в файл. Для этого, чтобы использовать механизм хеша TPM, необходимо создать объект хеша, выполнить хеширование открытого ключа, используя алгоритм SHA-1, а затем производить считывание хеша из объекта TPM.

Алгоритм данного модуля представлен на рисунке13.

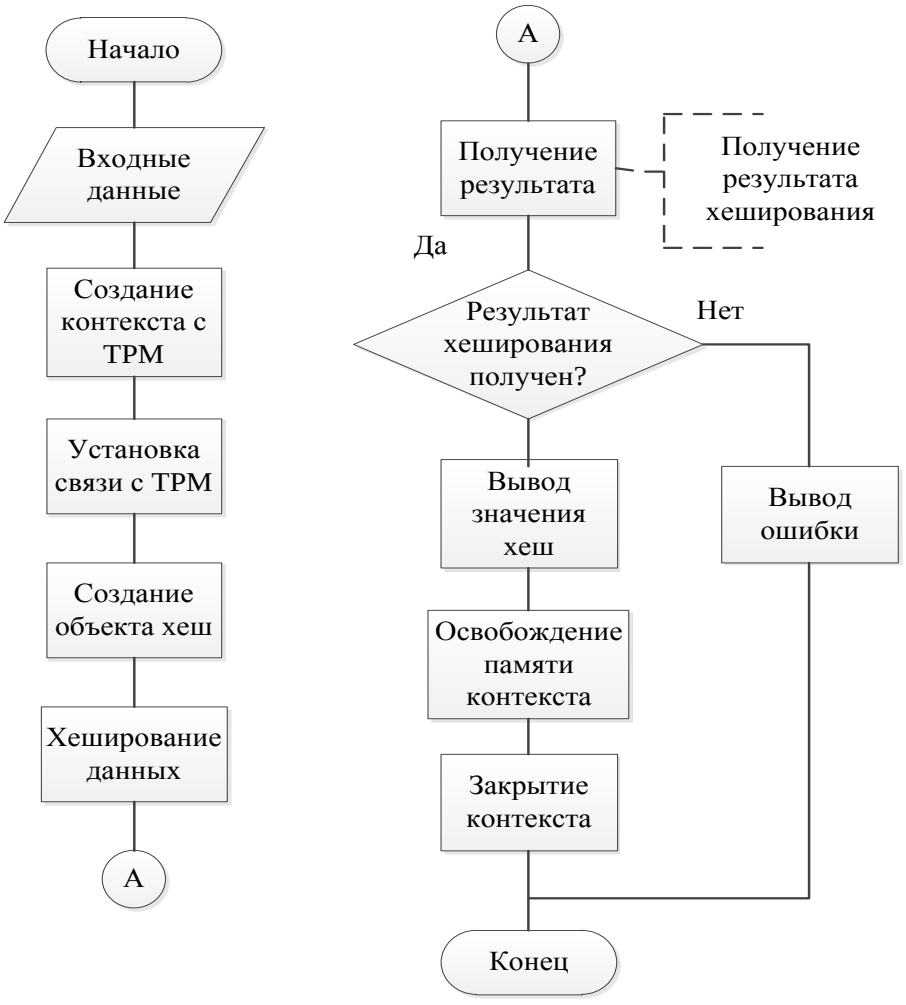


Рисунок 13 – Алгоритм модуля хеширования данных

5.1.2 Алгоритм модуля создания энергонезависимой памяти (NVRAM).

Данный модуль позволяет создавать NVRAM указанного размера.

Алгоритм разрабатываемого модуля представлен на рисунке 14.

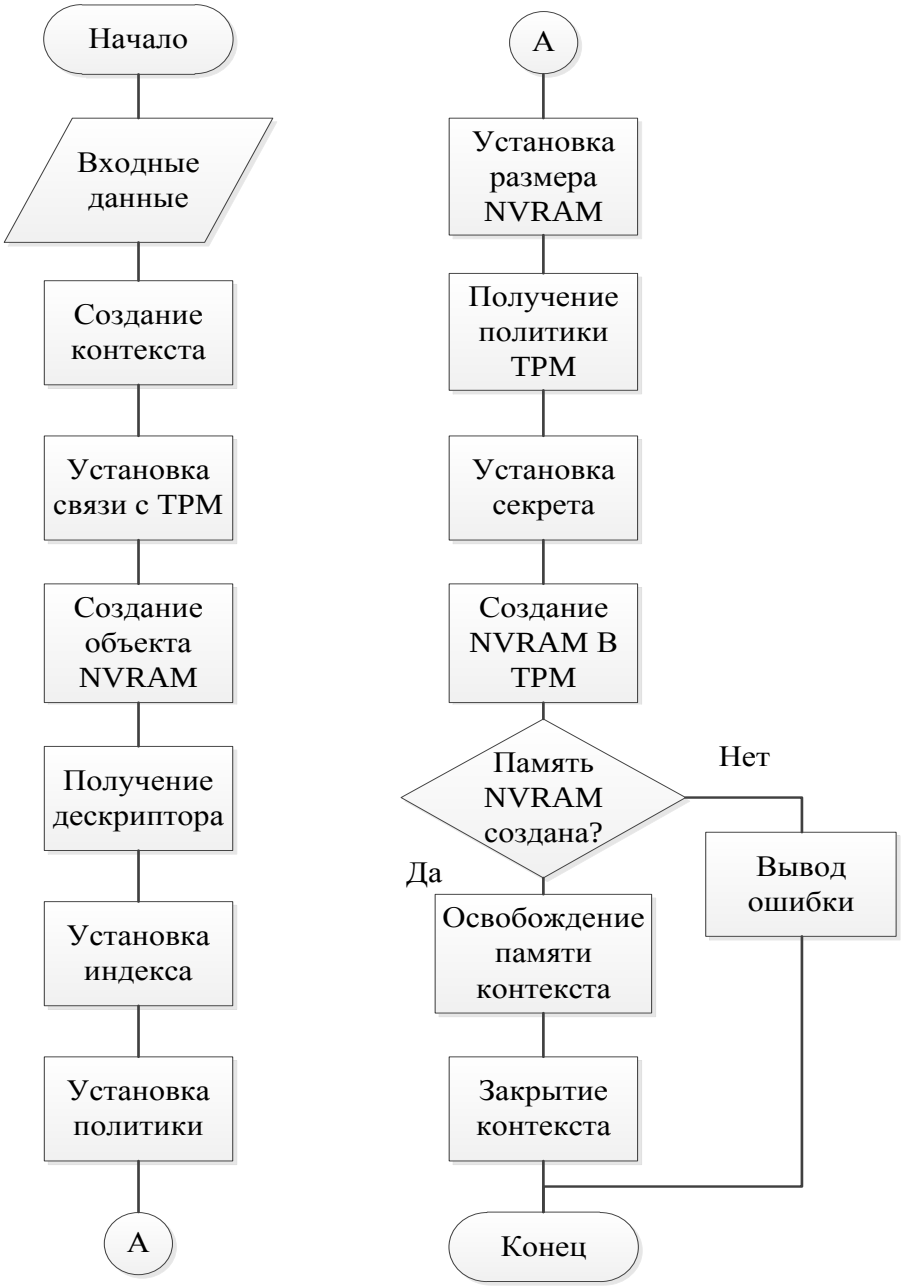


Рисунок 14 – Алгоритм модуля создания NVRAM

5.1.3 Алгоритм модуля записи в энергонезависимую память (NVRAM).

Данный модуль позволяет записывать необходимые данные указанного размера в NVRAM.

Алгоритм разрабатываемого модуля приведен на рисунке 15.

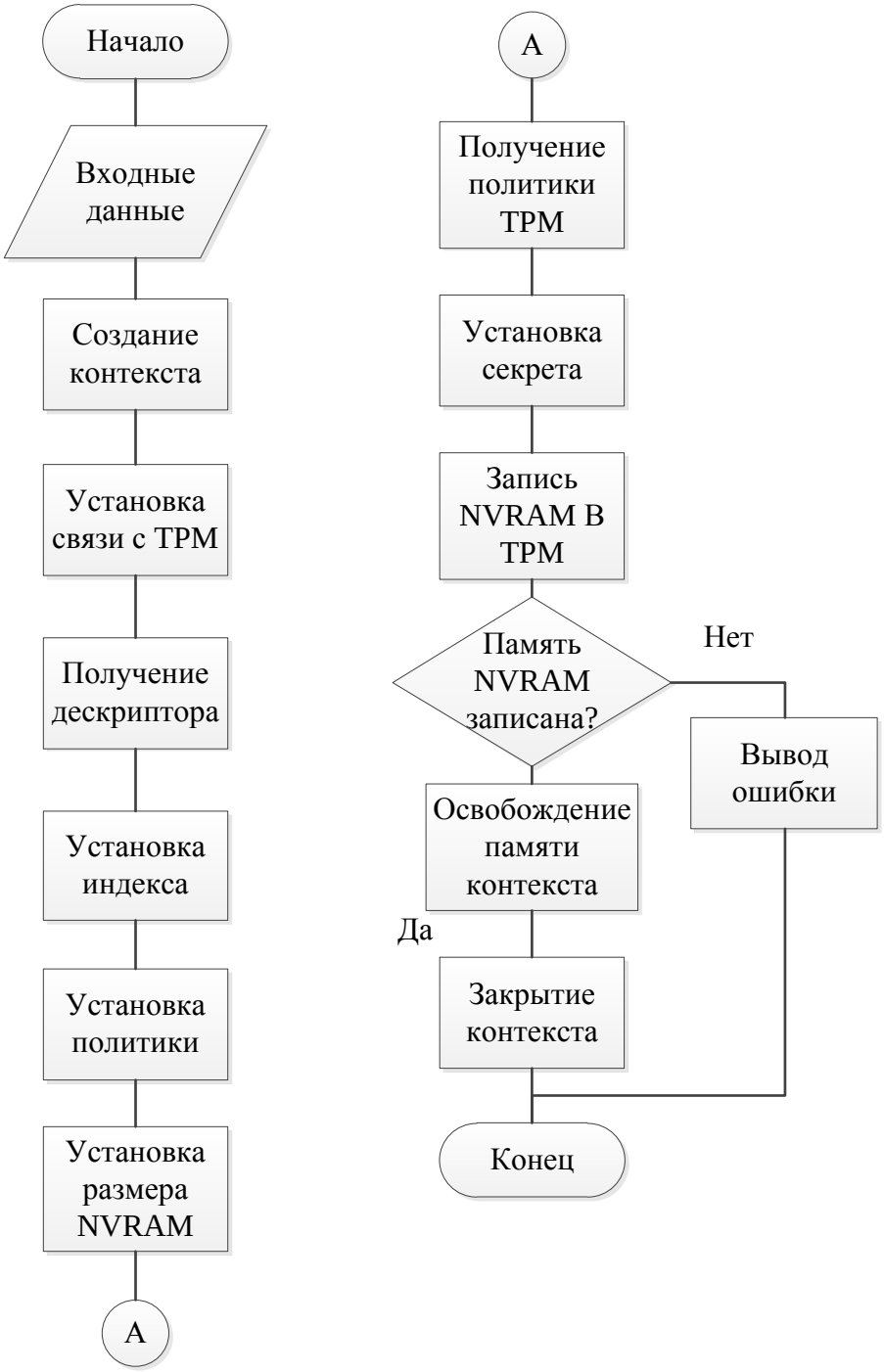


Рисунок 15 – Алгоритм модуля записи NVRAM

5.1.4 Алгоритм модуля чтения из энергонезависимой памяти (NVRAM). Данный модуль позволяет читать данные указанного размера из NVRAM. Алгоритм разрабатываемого модуля приведен на рисунке 16.

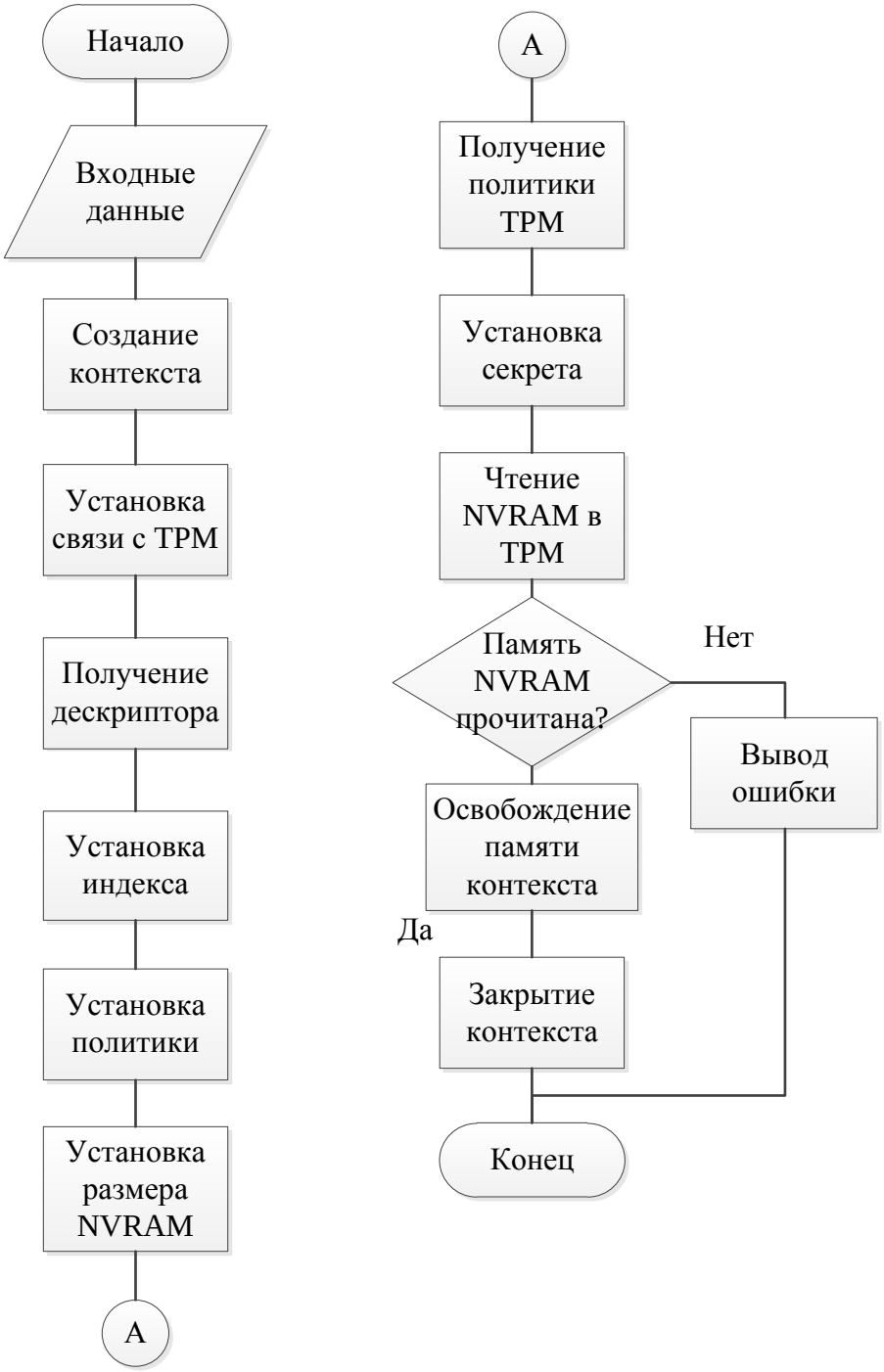


Рисунок 16 – Алгоритм модуля чтения NVRAM

5.1.5 Алгоритм модуля удаления энергонезависимой памяти (NVRAM).

Данный модуль освобождает NVRAM в TPM.

Алгоритм программы приведен на рисунке 17.

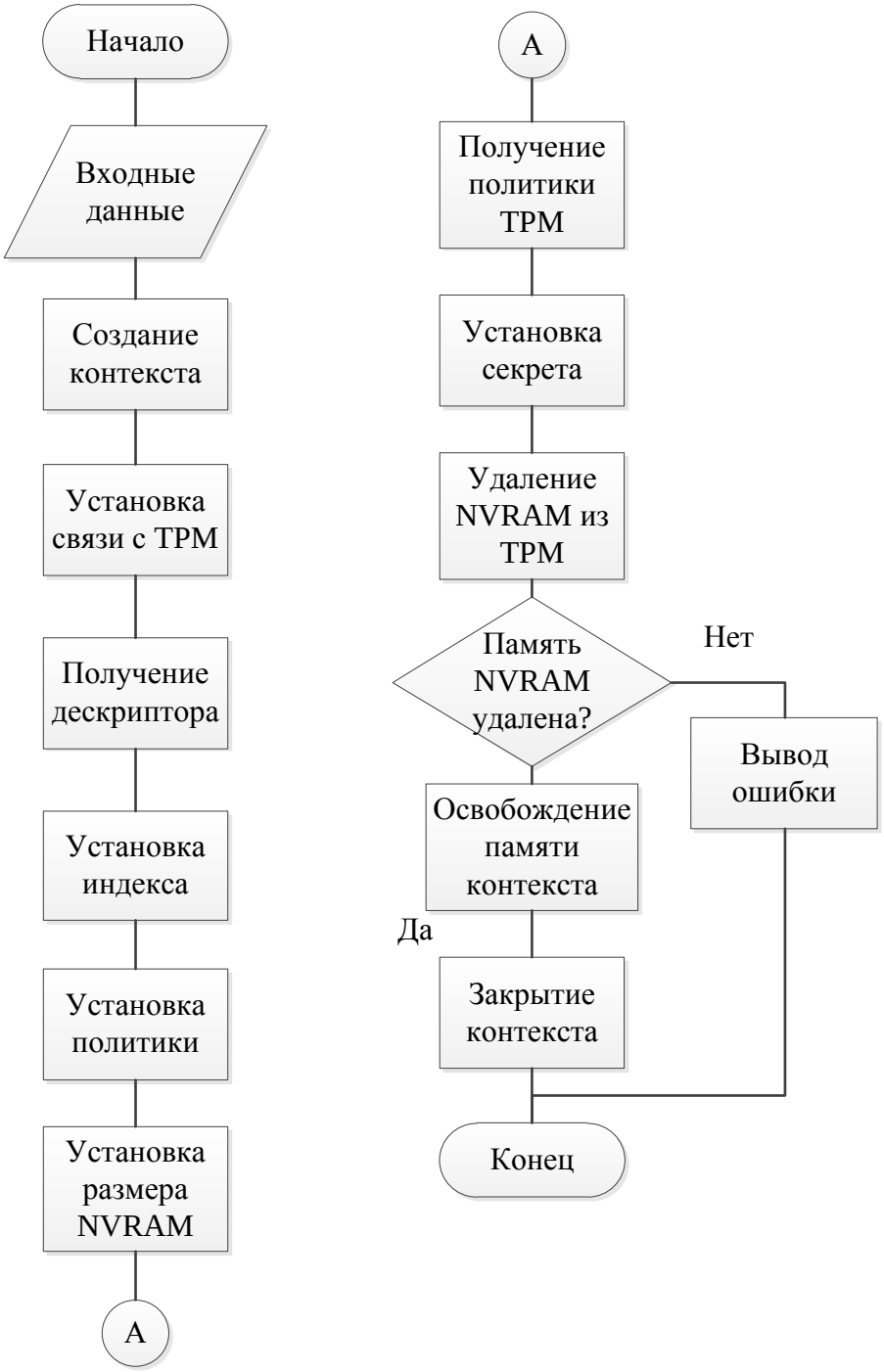


Рисунок 17 – Алгоритм модуля удаления NVRAM

5.1.6 Алгоритм модуля шифрования данных.

Данный модуль осуществляет шифрование данных, используя алгоритм RSA, и запись зашифрованных связанных данных в файл.

Для того, чтобы данные шифровались корректно необходимо, чтобы был загружен в память корневой ключ хранения SRK.

Алгоритм разрабатываемого модуля представлен на рисунке 18.

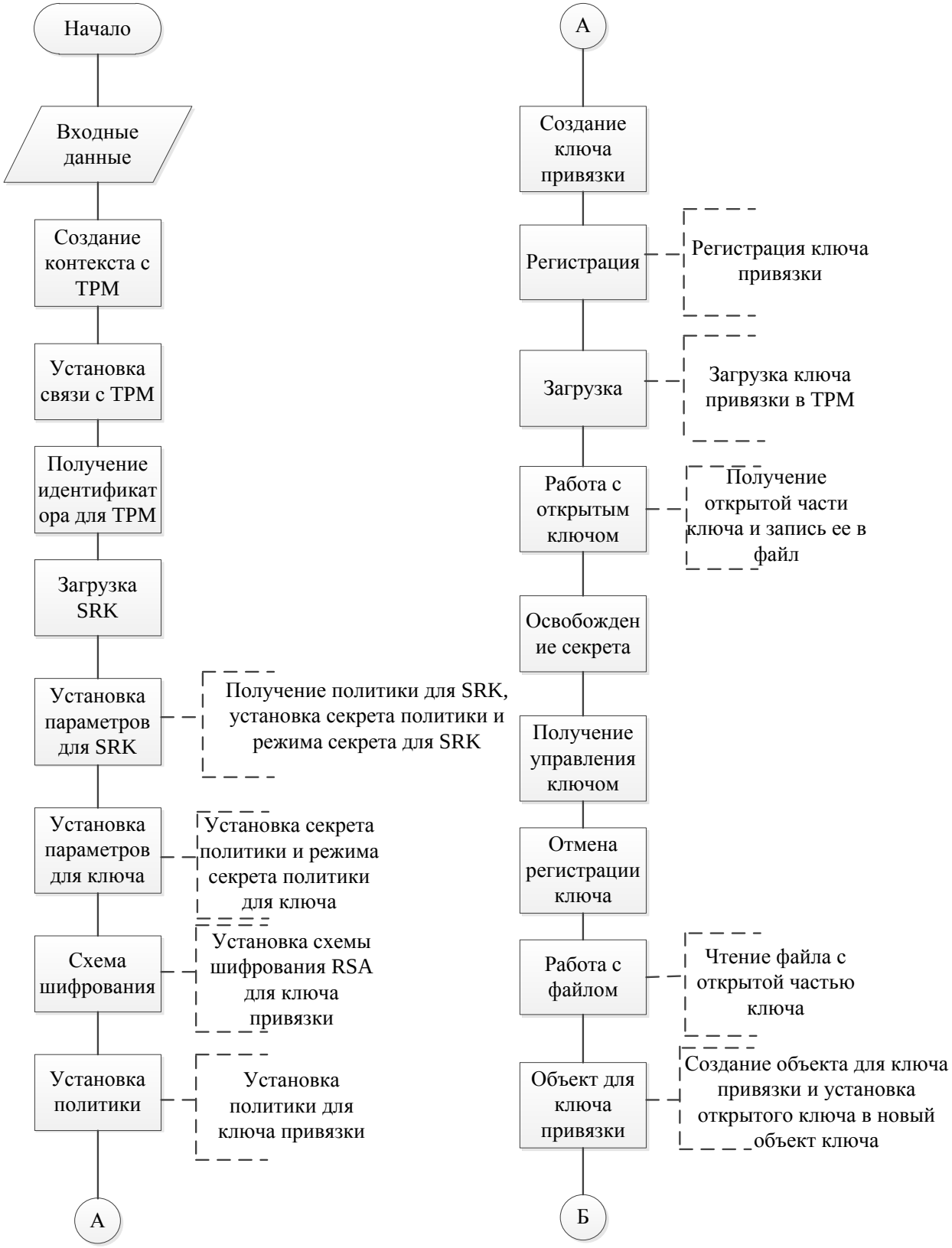


Рисунок 18 – Алгоритм модуля шифрования данных, лист 1

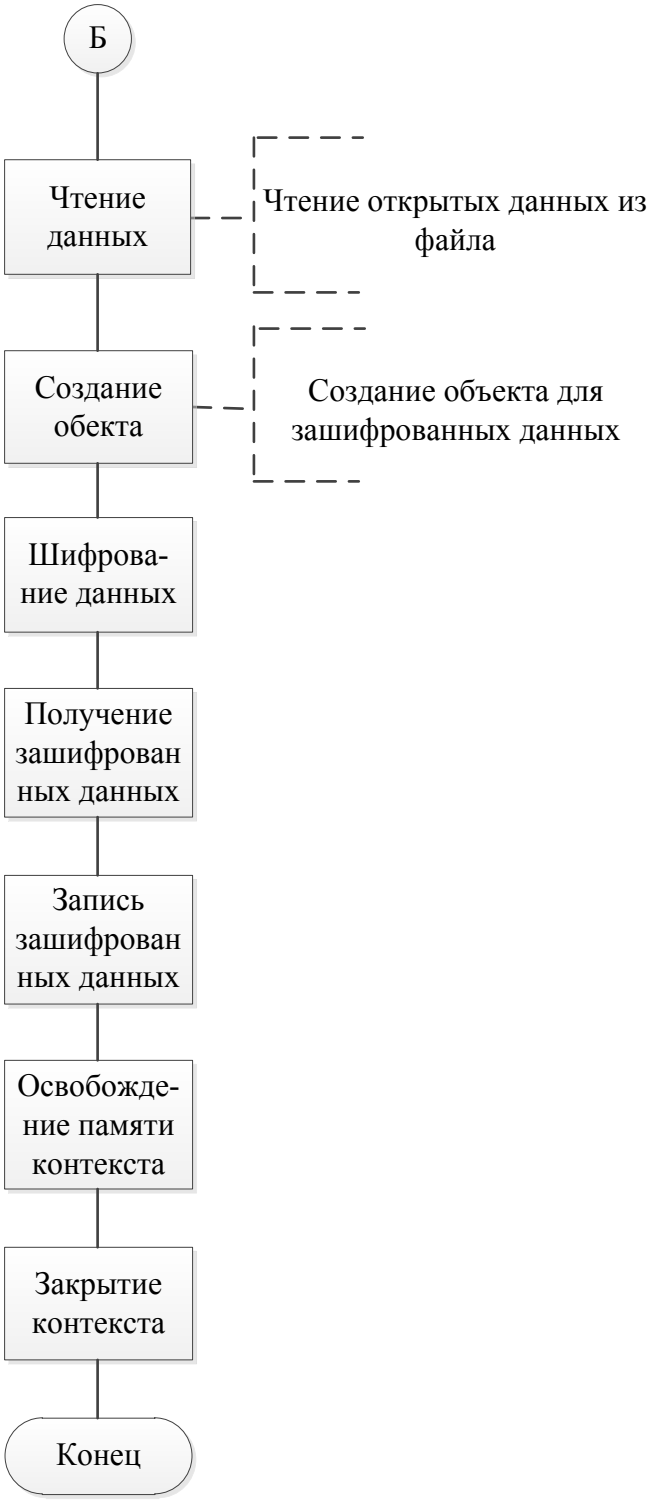


Рисунок 18 – Алгоритм модуля шифрования данных, лист 2

5.1.7 Алгоритм модуля расшифрования данных.

Данный модуль осуществляет расшифрование зашифрованных данных, используя алгоритм RSA, и запись расшифрованных данных в файл.

Для этого необходим также ключ SRK.

Алгоритм разрабатываемого модуля приведен на рисунке 19.

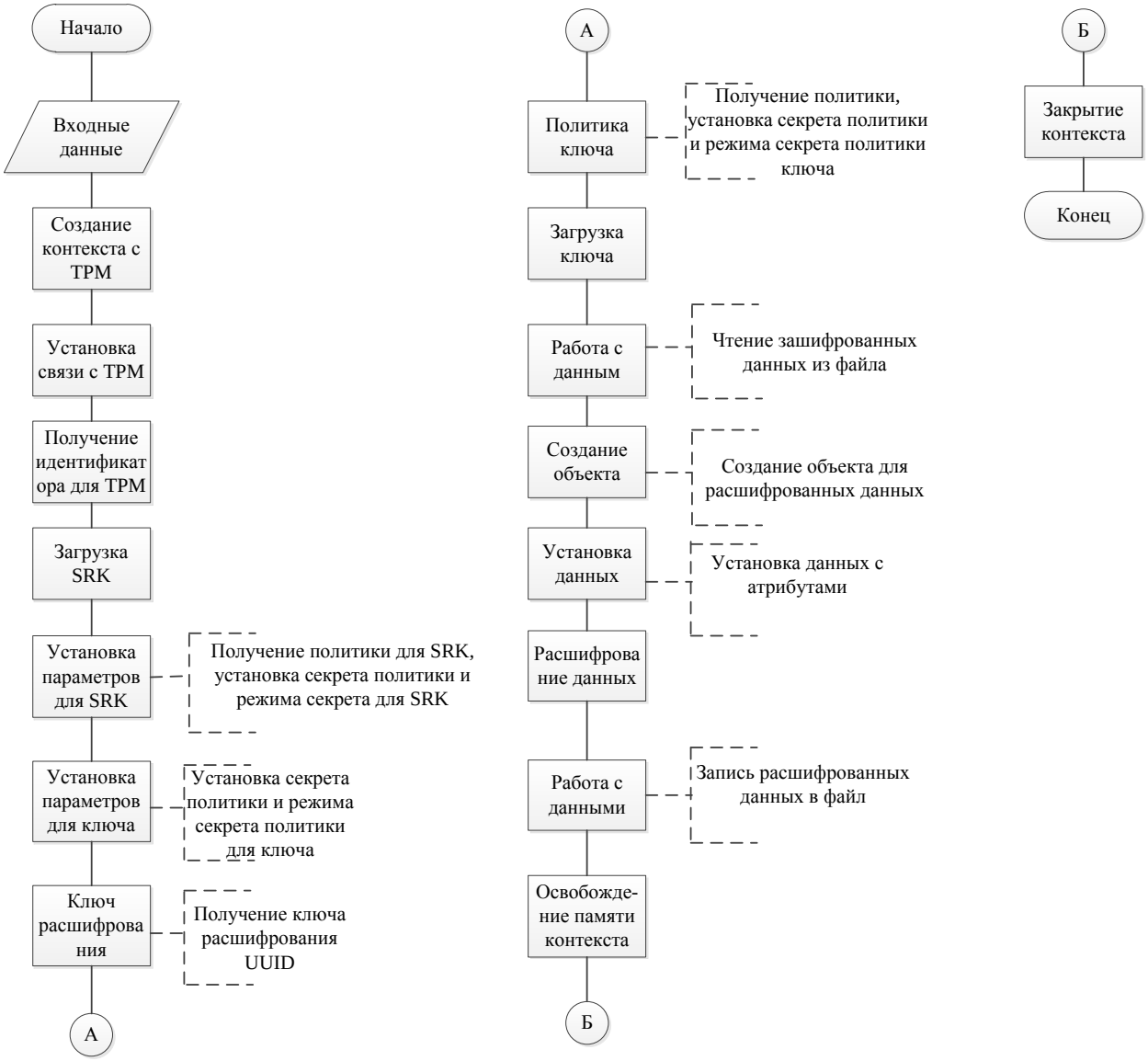


Рисунок 19 – Алгоритм модуля расшифрования данных

5.1.8 Алгоритм модуля генерации случайных чисел.

Данный модуль позволяет осуществлять рандомную выработку чисел размер, которых не должен превышать 4096 байт.

Алгоритм разрабатываемого модуля приведен на рисунке 20.

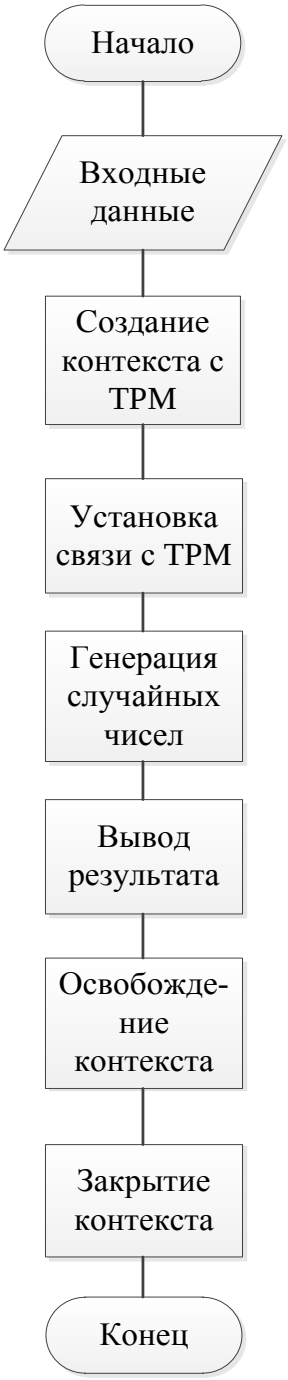


Рисунок 20 – Модуль генерации случайных чисел

5.1.9 Алгоритм модулей чтения, сброса, смены PCR.

Модуль чтения PCR осуществляет чтение и вывод на экран текущего значения PCR.

Модуль сброса PCR осуществляет сброс значений PCR.

Модуль смены PCR осуществляет смену PCR. Алгоритм разрабатываемых модулей представлен на рисунке 21.

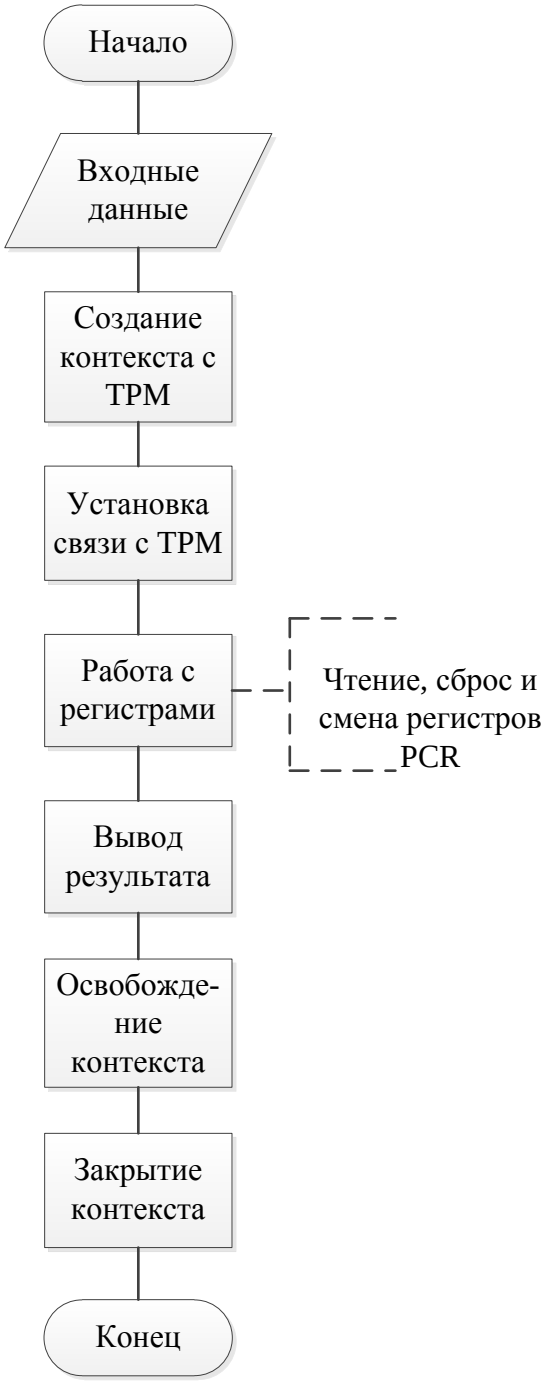


Рисунок 21 – Алгоритм модулей чтения, сброса и смены PCR

5.1.10 Алгоритм модуля опечатывания данных.

Данный модуль позволяет опечатывать данные и записывать опечатанные данные в файл. Алгоритм модуля приведен на рисунке 22.

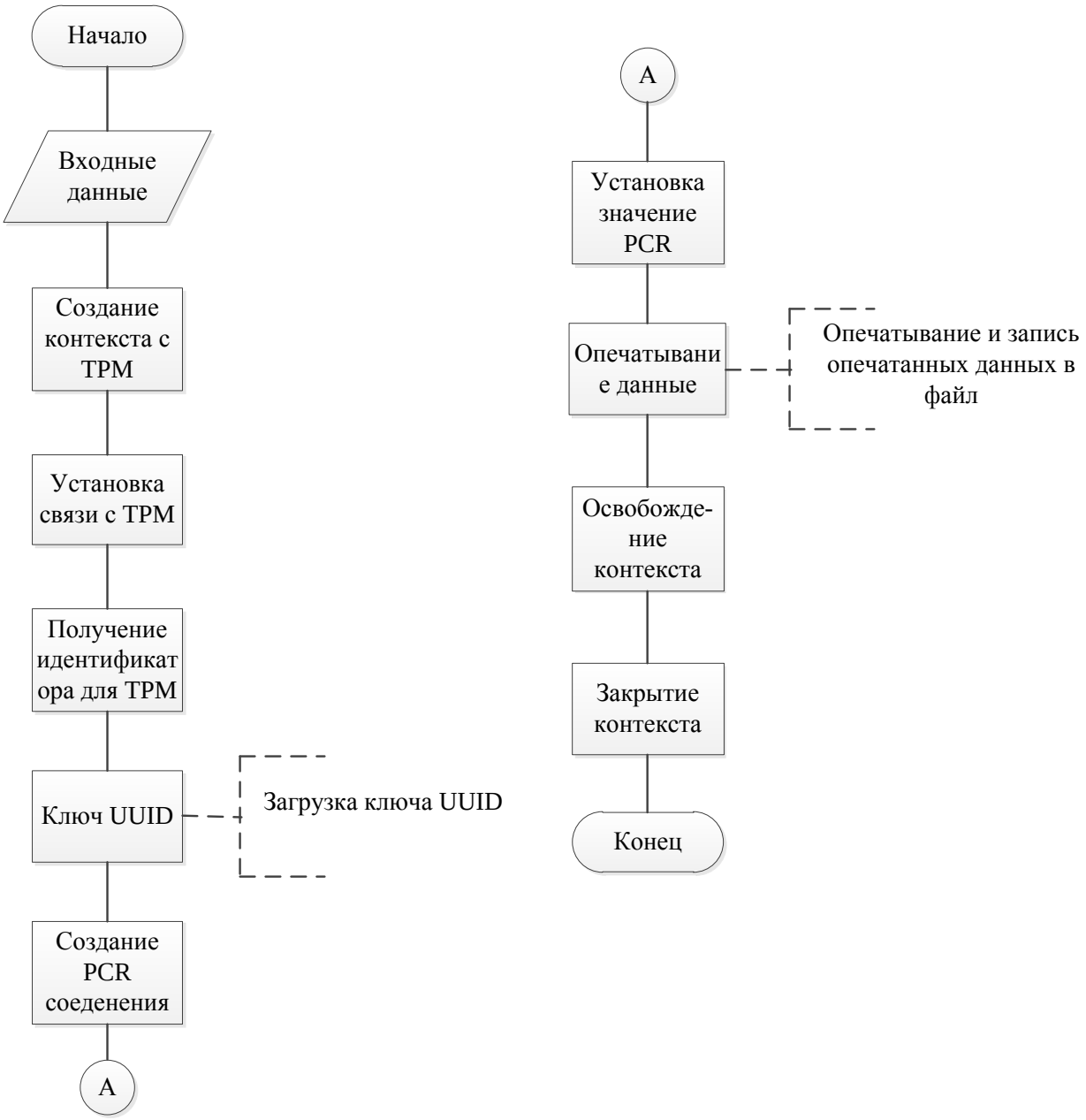


Рисунок 22 – Алгоритм модуля опечатывания данных

5.1.11 Алгоритм модуля распечатывания данных.

Данный модуль позволяет осуществлять распечатывание данных, а также запись распечатанных данных в файл. Алгоритм модуля приведен на рисунке 23.

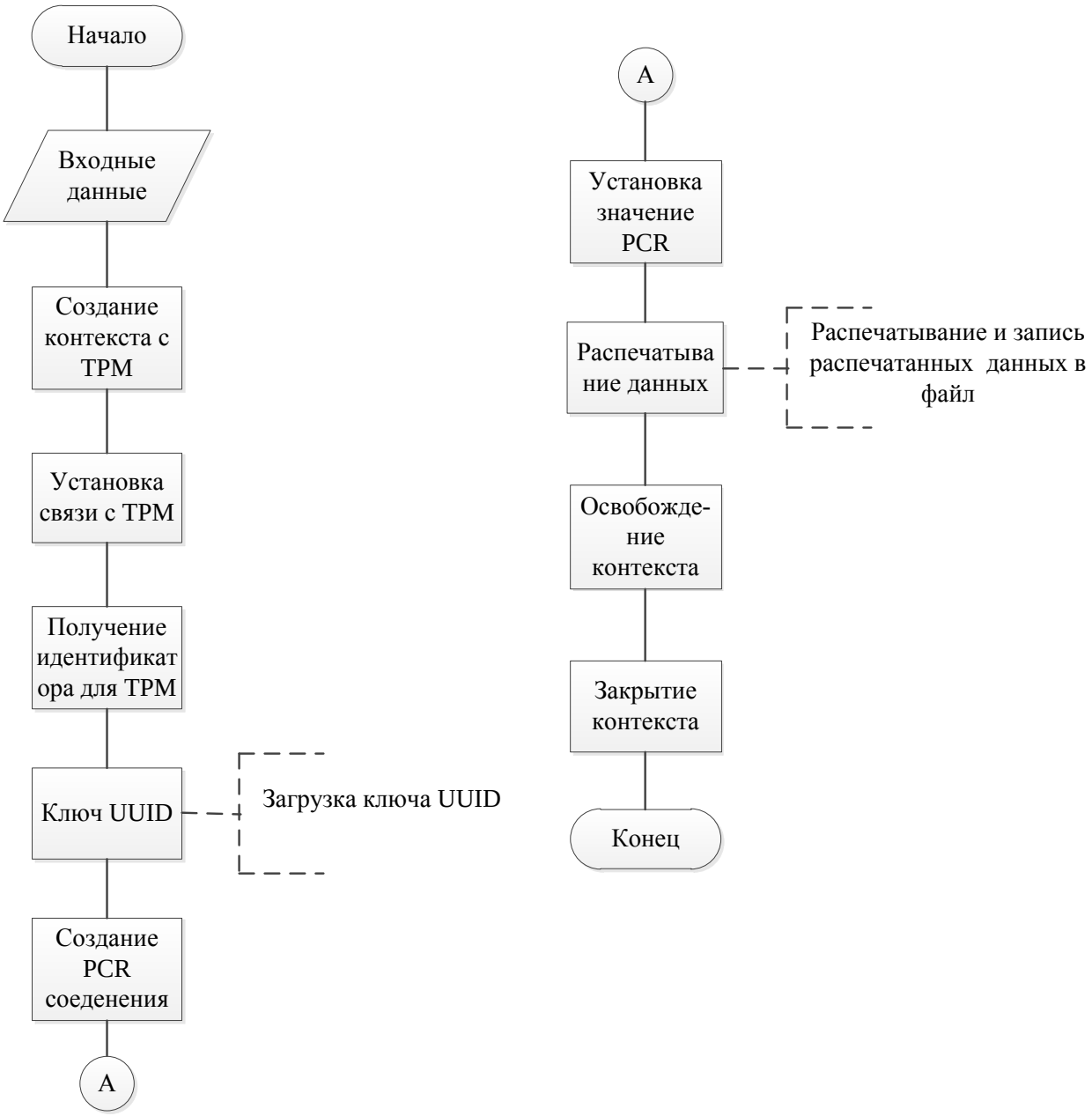


Рисунок 23 – Алгоритм модуля распечатывания данных

5.1.12 Алгоритм модуля создания ключа подписи.

Данный модуль позволяет осуществлять генерацию ключа подписи, при этом используя алгоритм RSA, а также позволяет регистрировать и загружать ключ в TPM. Алгоритм модуля представлен на рисунке 24.

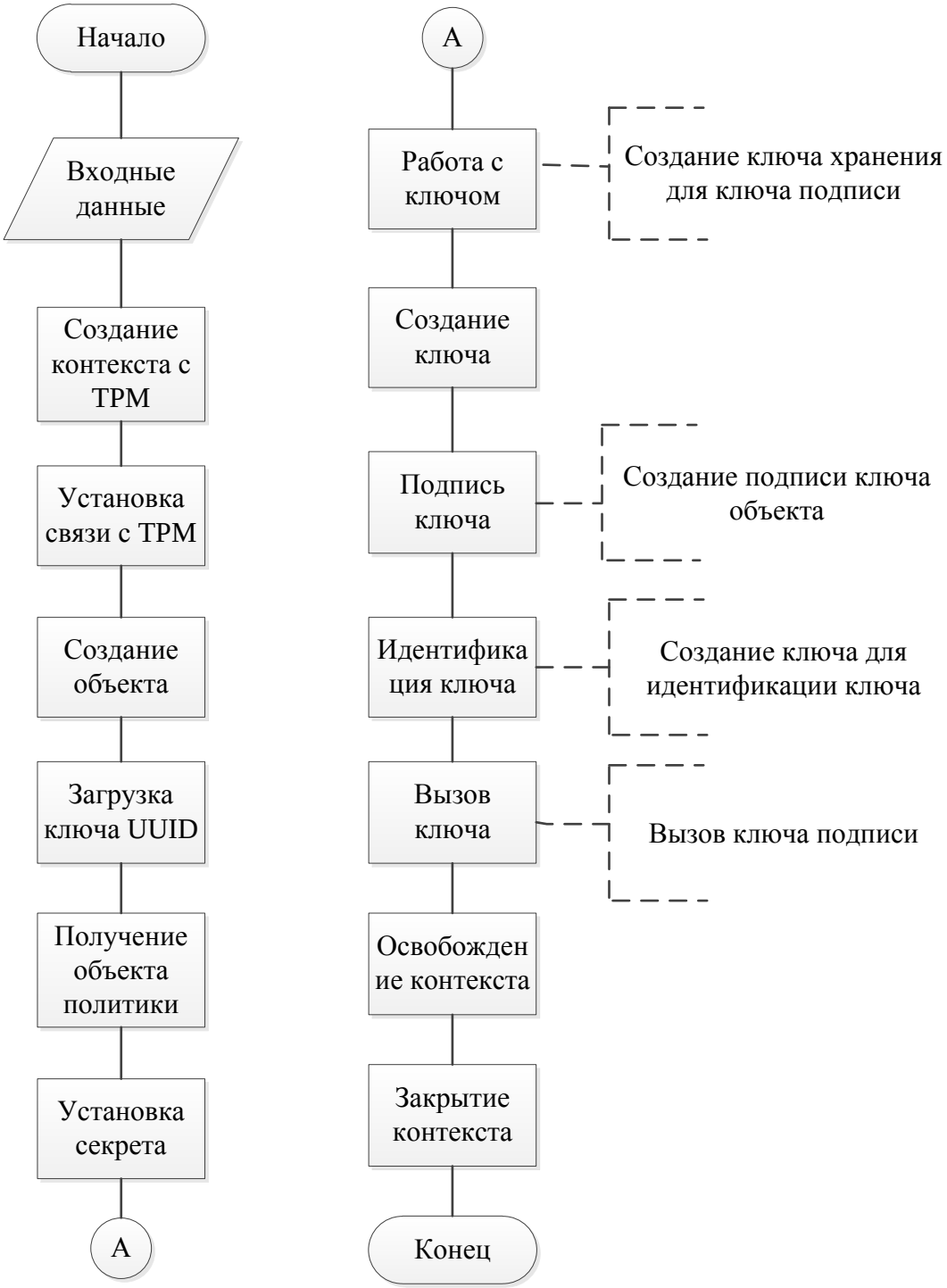


Рисунок 24 – Алгоритм модуля создания ключа подписи

5.1.13 Алгоритм модуля создания ключа привязки.

Данный модуль позволяет создавать ключ привязки, который осуществляет генерацию ключа привязки, используя алгоритм RSA, а также регистрировать и загружать ключ в TPM. Алгоритм модуля представлен на рисунке 25.

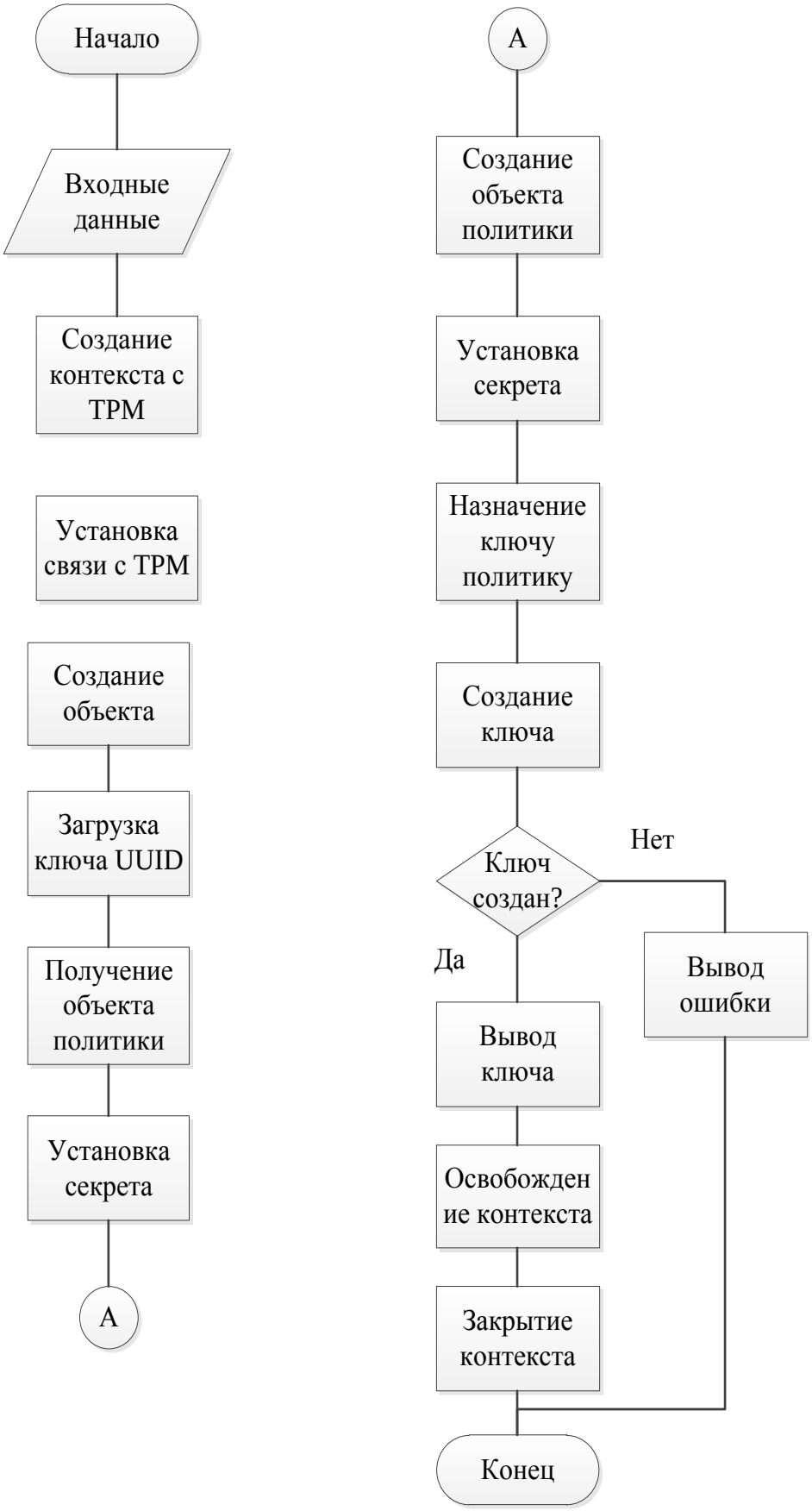


Рисунок 24 – Алгоритм модуля создания ключа привязки

5.1.14 Алгоритм модуля удаления ключа.

Данный модуль позволяет осуществлять удаления ключей, которые находятся в TPM. Алгоритм модуля приведен на рисунке 26.

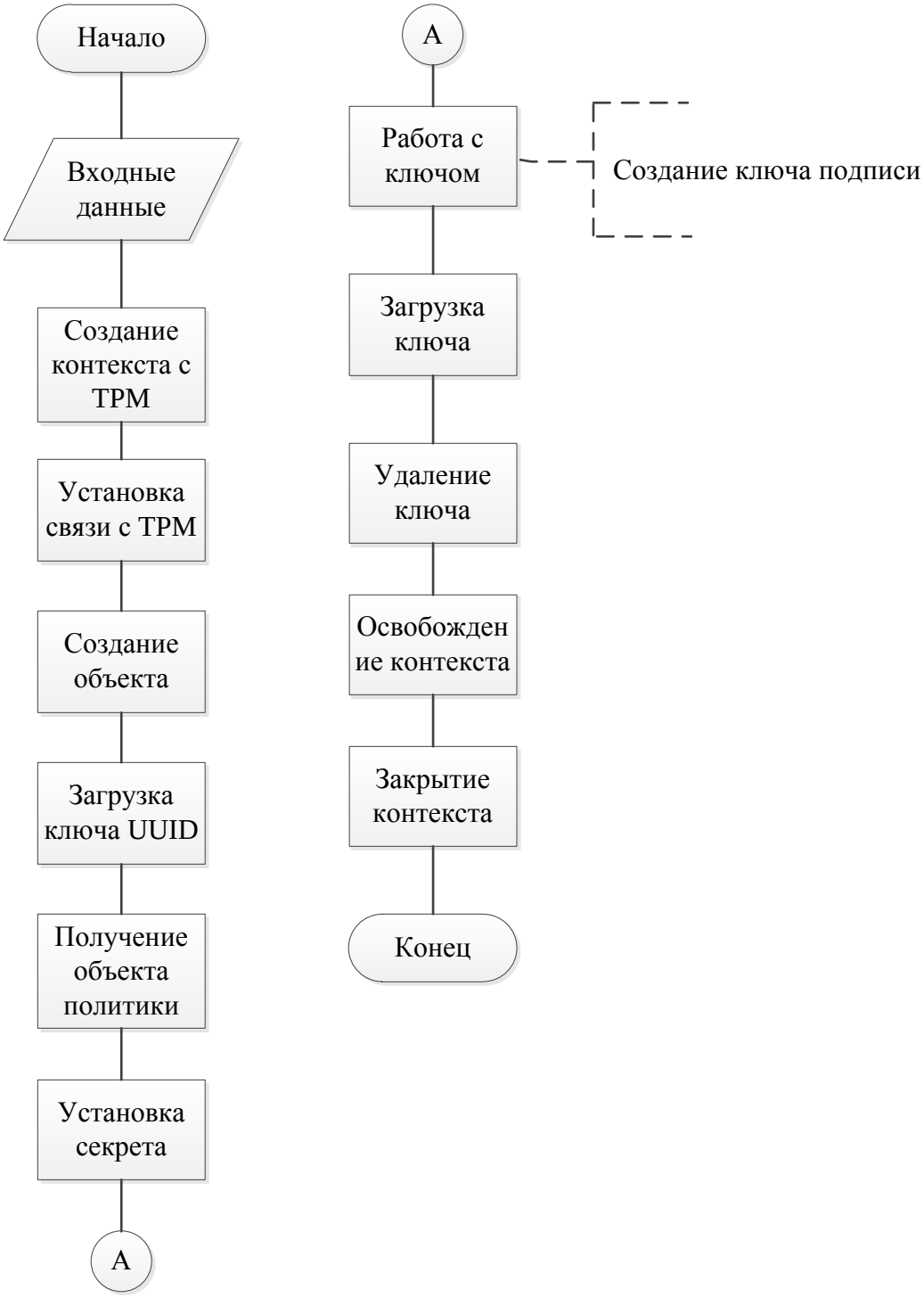


Рисунок 26 – Алгоритм модуля удаления ключа

5.1.15 Алгоритм модуля создание ключа подтверждения подлинности AIK. Данный ключ RSA должен обладать длинной 2048 бит. Он используется в качестве ключа подписи. Алгоритм данного модуля представлен на рисунке 27.

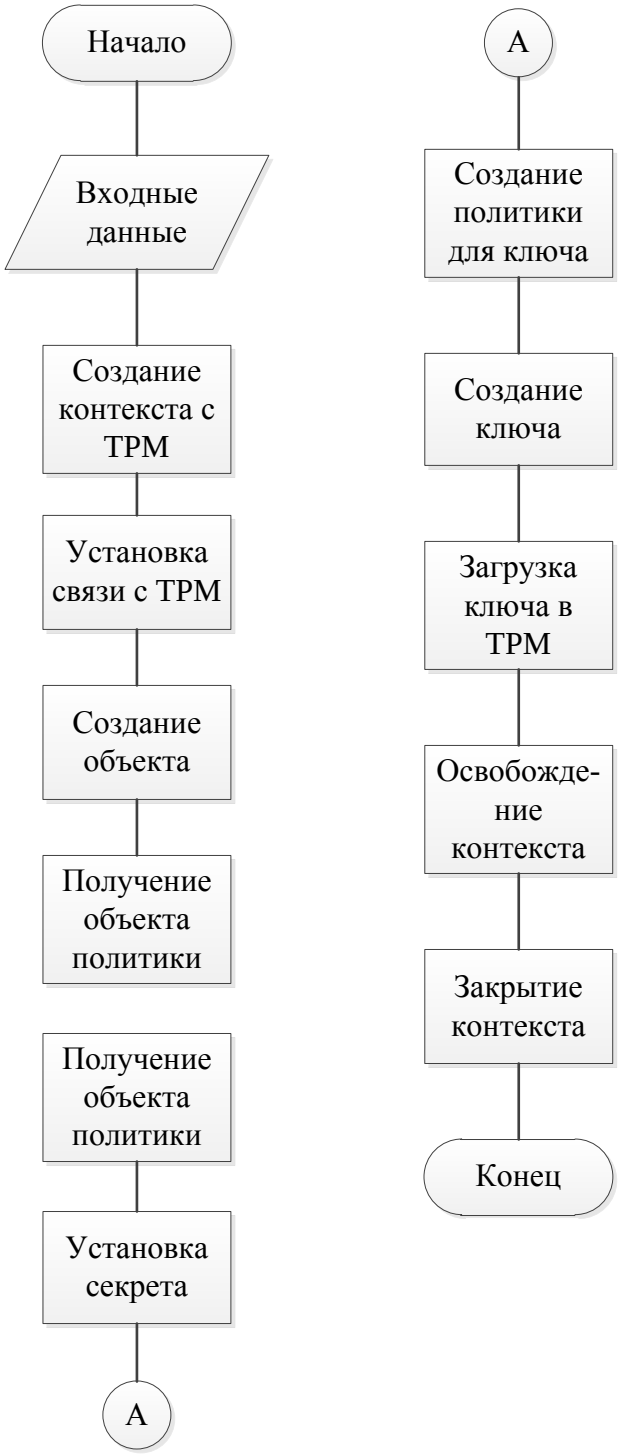


Рисунок 27 – Алгоритм модуля создания ключа подтверждения подлинности AIK

Далее, исходя из выше описанных алгоритмов, составлен обобщенный алгоритм работы программного средства тестирования и управления модулем доверенной платформы. Алгоритм приведен на рисунке 28.

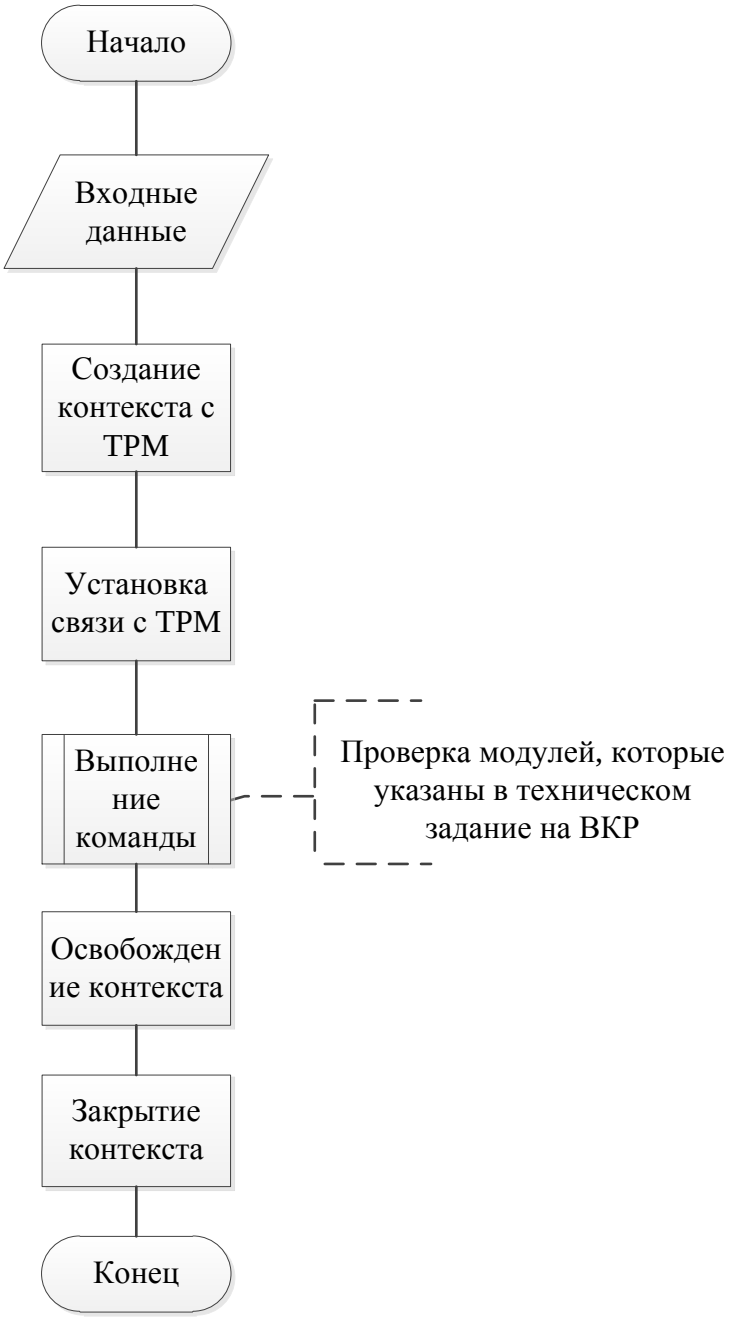


Рисунок 28 – Алгоритм работы программного средства тестирования и управления модулем

Таким образом, алгоритмы модулей программного средства тестирования были выполнены в полном объеме.

5.2 Разработка программного средства

В данном вопросе были разработаны модули программного средства, которые приведены в ТЗ на ВКР, такие как:

- модуль создания ключа подписи, который должен осуществлять генерацию ключа подписи, используя алгоритм RSA, регистрацию и загрузку ключа в TPM;
- модуль создания ключа привязки, который должен осуществлять генерацию ключа привязки, используя алгоритм RSA, регистрацию и загрузку ключа в TPM;
- модуль удаления ключа, который должен осуществлять удаление ключа из TPM;
- модуль шифрования данных, который должен осуществлять шифрование данных, используя алгоритм RSA, и запись зашифрованных связанных данных в файл;
- модуль расшифрования данных, который должен осуществлять расшифрование зашифрованных данных, используя алгоритм RSA, и запись расшифрованных данных в файл;
- модуль опечатывания данных, который должен осуществлять опечатывание данных и запись опечатанных данных в файл;
- модуль распечатывания данных, который должен осуществлять распечатывание данных и запись распечатанных данных в файл;
- модуль хеширования данных, который должен осуществлять хеширование открытого ключа, используя алгоритм SHA-1;
- модуль генерации случайных чисел, который должен осуществлять рандомную выработку чисел размером, не более 4096 байт;
- модуль подписи данных, который должен осуществлять подпись данных, используя алгоритм хеширования SHA1 и сохранять подпись в файле;

– модуль подтверждения подлинности данных, который должен хешировать исходный файл, получать ключ подписи, получать подпись, используя алгоритм SHA1, проверять подлинность подписи;

– модуль создания NVRAM, который должен создавать NVRAM указанного размера;

– модуль записи в NVRAM, который должен записывать необходимые данные указанного размера в NVRAM;

– модуль чтения из NVRAM, который должен читать данные указанного размера из NVRAM;

– модуль удаления NVRAM, который должен освобождать NVRAM;

– модуль чтения PCR, который должен читать и выводить на экран текущие значения PCR;

– модуль сброса PCR, который должен осуществить сброс значений PCR;

– модуль смены PCR, который должен осуществлять смену PCR;

– модуль создания ключа подтверждения подлинности, который должен создавать ключ, используя алгоритм RSA.

Текст программ разработанных модулей приведен в приложение А.

Разработка программного средства выполнялась на языке программирования С. Команды программного средства тестирования и управления модулем доверенной платформы приведены в таблице 3.

Чтобы программы разрабатываемого модуля работали корректно, необходимо, чтобы они включали следующие библиотеки:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<tss/platform.h>
#include<tss/tss_defines.h>
#include<tss/tss_typedef.h>
#include<tss/tss_structs.h>
#include<tss/tspi.h>
#include<trousers/trousers.h>
#include<tss/tss_error.h>
```

Для вывода на экран результатов должны присутствовать следующие строки программного кода:

```
#define DEBUGTPM 1
#define DBGTPM(messagetpm, tResulttpm) { if(DEBUGTPM) printf("(Line%d, %s) %s returned 0x%08x. %s.\n",__LINE__ ,__func__ , messagetpm, tResulttpm,(char *)Trspi_Error_String(tResulttpm));}
```

Таблица 3 – Команды программного средства тестирования и управления модулем доверенной платформы

Команда	Описание
Команды тестирования и управления криптографическим процессором	
tpm_createsignkey	Создание, регистрация и загрузка ключа подписи TPM
tpm_createbindkey	Создание, регистрация и загрузка ключа привязки в TPM
tpm_binddata	Шифрование данных
tpm_unbinddata	Расшифрование данных
tpm_sealdata	Опечатывание данных
tpm_unsealdata	Распечатывание данных
tpm_evictkeys	Удаление ключа
tpm_hash	Установка и получение значения хеш – функции
tpm_signdata	Подпись данных
tpm_verifysign	Проверка подлинности подписанных данных
tpm_getrandom	Генерация случайны чисел
Команда тестирования и управления энергонезависимой памяти	
tpm_createnvram	Создание области энергонезависимой памяти NVRAM (хранит ключ подтверждения подлинности, корневой ключ хранения, данные авторизации, флаги состояния)

Продолжение таблицы 3

Команда	Описание
tpm_writenvram	Запись в энергонезависимую память
tpm_readnvram	Чтение из энергонезависимой памяти
tpm_destroynvram	Удаление энергонезависимой памяти
Команды тестирования и управления энергонезависимой памяти	
tpm_readpcr	Чтение регистров конфигурации платформы PCR (содержит данные о целостности платформы)
tpm_resetpcr	Сброс регистров конфигурации платформы
tpm_extendpcr	Смена регистров конфигурации платформы
tpm_createaik	Создание ключа подтверждения подлинности AIK (ключ RSA длинной 2048 бит. Используется в качестве ключа подписи)

5.3 Отладка программного средства

Отладка программного средства тестирования и управления модулем доверенной платформы, функционирующего в средствах вычислительной техники, предназначенных для обработки информации ограниченного доступа осуществлялась с помощью отладчика GDB.

GDB позволяет в определенной степени осуществлять контроль за выполняемой программой, то есть определить, что происходит в процессе ее выполнения.

GDB позволяет выполнять действия четырех основных типов, чтобы выявить ошибку в программе:

- остановка программы при указанных условиях;

- исследование кода, когда отладка программы остановилась;

- изменение программы для устранения ошибки и продолжения нахождения других ошибок;

- начало выполнение программы, задав все параметры, которые могут повлиять на ее поведение.

Для того чтобы отладка разрабатываемого программного средства выполнялась корректно, были выполнены следующие действия:

- запуск TPM эмулятора;

- запуск программного стека;

- компиляция программного средства для отладки с помощью команды `gcc.file.c –g file –ltspi –wall`;

- запуск отладчика с помощью команды `gdb.file`.

Параметр – `o` позволяет записывать вывод сборки в выходной файл.

Параметр – `Wall` позволяет выводить все предупреждающие сообщения компилятора.

Параметр – `g` используется при компиляции исходных файлов для отладки, а также для компоновщика, чтобы имелась возможность применяться специальные версии стандартной библиотеки С, обеспечивающие поддержку режима отладки в библиотечных функциях. [15]

Параметр – `ltspi` позволяет подключить библиотеки программного стека TrouSerS.

Отладчик включает в себя команды, которые приведены в таблице 4. [15]

Таблица 4 – Команды отладчика GDB

Команды	Описание команд
- help	Предоставляет интерактивную систему помощи и полное справочное руководство
- run	Выполнение программы
- backtrace	Определение маршрута, приведенного к ошибке
- print	Отображает содержимое переменных и других выражений

Продолжение таблицы 4

Команды	Описание команд
- list	Вывод на экран исходного текста программы, расположенного рядом с текущей позицией
- break	Установка точки останова на указанной строке.
- count	Продолжение выполнения программы
- display	Задает автоматическое отображение массива при каждой остановке программы в точке останова
- commands	Указывает, какие команды отладчика выполнять при попадании в точку останова
- info	Просмотр включенных точек останова
- disable	Отключение или удаление точек останова
- quit	Завершение работы отладчика GDB

Таким образом, отладка программного средства тестирования и управления модулем доверенной платформы, функционирующего в средствах вычислительной техники, предназначенных для обработки информации ограниченного доступа была выполнена в полном объеме и производилась по всем правилам, которые представлены выше.

6 Тестирование модуля доверенной платформы, с использованием разработанного программного средства

В данном разделе было произведено тестирование разрабатываемых модулей. Если модуль был разработан корректно и в нем выполняются все требования, то на экран выводится сообщение вида «Success», иначе выводится сообщение об ошибке.

Перед тестированием модулей, происходила его отладка с помощью отладчика GDB. На этом этапе были устранены все ошибки и недочеты в коде программ разрабатываемых модулей. Результаты тестирования программного модуля приведены ниже.

Результат тестирования модуля создания, регистрации и загрузки ключа подписи TPM приведен на рисунке 29.

```
[root@localhost diplom]# ./tpm_createsignkey
(Line41, main) Create a context returned 0x00000000. Success.
(Line43, main) Connect to TPM returned 0x00000000. Success.
(Line45, main) Get TPM handle returned 0x00000000. Success.
(Line47, main) Get SRK handle returned 0x00000000. Success.
(Line49, main) Get SRK Policy returned 0x00000000. Success.
(Line51, main) Tspi_Policy_SetSecret returned 0x00000000. Success.
(Line55, main) Create policy object returned 0x00000000. Success.
(Line58, main) Set Policy secret returned 0x00000000. Success.
(Line65, main) Create the key object returned 0x00000000. Success.
(Line70, main) Set the keys's padding type returned 0x00000000. Success.
(Line72, main) Assign the key's policy to the key returned 0x00000000. Success.
Creating the key could take a while
(Line76, main) Asking TPM to create the key returned 0x00000000. Success.
(Line83, main) Registering the key for later retrieval returned 0x00000000. Success.
Registering the key blob for later retrieval
(Line88, main) Loading key in TPM returned 0x00000000. Success.
(Line91, main) Get Public portion of key returned 0x00000000. Success.
Finished writing SigningKey.pub
(Line102, main) Policy flush secret returned 0x00000000. Success.
(Line108, main) Get key handle returned 0x00000000. Success.
Unregistering key
(Line114, main) Unregister key returned 0x00000000. Success.
(Line117, main) Tspi Context Free Memory returned 0x00000000. Success.
(Line119, main) Tspi Context Close returned 0x00000000. Success.
```

Рисунок 29 – Результат тестирования модуля создания, регистрации и загрузки ключа TPM

Результат тестирования модуля создания, регистрации и загрузки ключа привязки TPM приведен на рисунке 30.

```
[root@localhost diplom]# ./tpm_createbindkey
(Line41, main) Create a context returned 0x00000000. Success.
(Line43, main) Connect to TPM returned 0x00000000. Success.
(Line47, main) Get TPM handle returned 0x00000000. Success.
(Line49, main) Get SRK handle returned 0x00000000. Success.
(Line51, main) Get SRK Policy returned 0x00000000. Success.
(Line53, main) Tspi_Policy_SetSecret returned 0x00000000. Success.
(Line58, main) Create a key policy object returned 0x00000000. Success.
(Line61, main) Set key policy object secret returned 0x00000000. Success.
(Line68, main) Create the key object returned 0x00000000. Success.
(Line73, main) Set the keys's padding type returned 0x00000000. Success.
(Line75, main) Assign the key's policy to the key returned 0x00000000. Success.
Creating the key could take a while
(Line79, main) Asking TPM to create the key returned 0x00000000. Success.
(Line86, main) Registering the key for later retrieval returned 0x00000000.
 Success.
Registering the key blob for later retrieval
(Line91, main) Loading key in TPM returned 0x00000000. Success.
(Line94, main) Get Public portion of key returned 0x00000000. Success.
Finished writing BackupESSBindKey.pub
(Line105, main) Policy flush secret returned 0x00000000. Success.
(Line108, main) Tspi Context Free Memory returned 0x00000000. Success.
(Line110, main) Tspi Context Close returned 0x00000000. Success.
```

Рисунок 30– Результат тестирования модуля создания, регистрации и загрузки ключа привязки TPM

Результат тестирования модуля шифрования данных приведен на рисунке 31.

```
[root@localhost diplom]# ./tpm_binddata
(Line41, main) Create a context returned 0x00000000. Success.
(Line43, main) Connect to TPM returned 0x00000000. Success.
(Line47, main) Get TPM handle returned 0x00000000. Success.
(Line49, main) Get SRK handle returned 0x00000000. Success.
(Line51, main) Get SRK Policy returned 0x00000000. Success.
(Line53, main) Tspi_Policy_SetSecret returned 0x00000000. Success.
(Line58, main) Create a key policy object returned 0x00000000. Success.
(Line61, main) Set key policy object secret returned 0x00000000. Success.
(Line68, main) Create the key object returned 0x00000000. Success.
(Line73, main) Set the keys's padding type returned 0x00000000. Success.
(Line75, main) Assign the key's policy to the key returned 0x00000000. Success.
Creating the key could take a while
(Line79, main) Asking TPM to create the key returned 0x00000000. Success.
(Line86, main) Registering the key for later retrieval returned 0x00000000. Success.
Registering the key blob for later retrieval
(Line91, main) Loading key in TPM returned 0x00000000. Success.
(Line94, main) Get Public portion of key returned 0x00000000. Success.
Finished writing BackupESSBindKey.pub
(Line105, main) Policy flush secret returned 0x00000000. Success.
(Line111, main) Get key handle returned 0x00000000. Success.
Unregistering key
(Line117, main) Unregister key returned 0x00000000. Success.
(Line131, main) Tspi_Context_CreateObject BindKey returned 0x00000000. Success.
(Line136, main) Set Public key into new key object returned 0x00000000. Success.
```

Рисунок 31 – Результат тестирования модуля шифрования данных

Результат тестирования модуля расшифрования данных приведен на рисунке 32.

```
[root@localhost diplom]# ./tpm_unbinddata
(Line41, main) Create a context returned 0x00000000. Success.
(Line43, main) Connect to TPM returned 0x00000000. Success.
(Line47, main) Get TPM handle returned 0x00000000. Success.
(Line49, main) Get SRK handle returned 0x00000000. Success.
(Line51, main) Get SRK Policy returned 0x00000000. Success.
(Line53, main) Tspi_Policy_SetSecret returned 0x00000000. Success.
(Line58, main) Create a key policy object returned 0x00000000. Success.
(Line61, main) Set key policy object secret returned 0x00000000. Success.
(Line68, main) Create the key object returned 0x00000000. Success.
(Line73, main) Set the keys's padding type returned 0x00000000. Success.
(Line75, main) Assign the key's policy to the key returned 0x00000000. Success.
Creating the key could take a while
(Line79, main) Asking TPM to create the key returned 0x00000000. Success.
(Line86, main) Registering the key for later retrieval returned 0x00000000. Success.
Registering the key blob for later retrieval
(Line91, main) Loading key in TPM returned 0x00000000. Success.
(Line94, main) Get Public portion of key returned 0x00000000. Success.
Finished writing BackupESSBindKey.pub
(Line105, main) Policy flush secret returned 0x00000000. Success.
(Line120, main) Tspi_Context_CreateObject BindKey returned 0x00000000. Success.
(Line125, main) Set Public key into new key object returned 0x00000000. Success.
```

Рисунок 32 – Результат тестирования модуля расшифрования данных

Результат тестирования модуля опечатывания данных приведен на рисунке 33.

```
[root@localhost diplom]# ./tpm_sealdata
(Line46, main) Create a context returned 0x00000000. Success.
(Line48, main) Connect to TPM returned 0x00000000. Success.
(Line50, main) Get TPM handle returned 0x00000000. Success.
(Line55, main) Get SRK handle returned 0x00000000. Success.
(Line57, main) Get SRK Policy returned 0x00000000. Success.
(Line59, main) Tspi_Policy_SetSecret returned 0x00000000. Success.
(Line66, main) Createing a PCR object returned 0x00000000. Success.
(Line69, main) Read the current value for PCR 8 returned 0x00000000. Success.
(Line71, main) Set the value read from PCR8 as sealing value of PCR8 returned 0x00000000. Success.
(Line75, main) Read the current value for PCR 9 returned 0x00000000. Success.
(Line77, main) Set the value read from PCR9 as sealing value of PCR9 returned 0x00000000. Success.
(Line83, main) Create data object for sealing returned 0x00000000. Success.
(Line88, main) Sealed my password, using the SRK key, to PCR 9 returned 0x00000000. Success.
(Line95, main) Get the encrypted sealed data returned 0x00000000. Success.
(Line108, main) Tspi Context Free Memory returned 0x00000000. Success.
(Line110, main) Tspi Context Close returned 0x00000000. Success.
```

Рисунок 33 – Результат тестирования модуля опечатывания данных

Результат тестирования модуля распечатывания данных приведен на рисунке 34.

```
[root@localhost diplom]# ./tpm_unsealdata
(Line37, main) Create a context
 returned 0x00000000. Success.
(Line39, main) Connect to TPM
 returned 0x00000000. Success.
(Line41, main) Get TPM handle
 returned 0x00000000. Success.
(Line46, main) Get SRK handle
 returned 0x00000000. Success.
(Line48, main) Get SRK Policy
 returned 0x00000000. Success.
(Line50, main) Tspi_Policy_SetSecret
 returned 0x00000000. Success.
(Line66, main) Createing data object for unsealing returned 0x00000000. Success.
(Line71, main) Set attrib data to hRetrieveData returned 0x00000000. Success.
(Line73, main) Unsealing data! returned 0x00000000. Success.
11, My Password
(Line79, main) Tspi Context Free Memory returned 0x00000000. Success.
(Line81, main) Tspi Context Close returned 0x00000000. Success.
```

Рисунок 34 – Результат тестирования модуля распечатывания данных

Результат тестирования модуля удаления ключа приведен на рисунке 35.

```
[root@localhost diplom]# ./tpm_evictkeys
Entering CreateOEAIK Temporary
Line46, main , Create Context, returned 0x00000000. Success.
Line48, main , Context Connect, returned 0x00000000. Success.
Line51, main , Get TPM Handle, returned 0x00000000. Success.
Line54, main , Get the SRK handle, returned 0x00000000. Success.
Line61, main , GetTPM policy, returned 0x00000000. Success.
Line63, main , Set TPM policy's secret to 123, returned 0x00000000.
Success.
Line66, main , Create a policy object for the key, returned
0x00000000. Success.
Line69, main , Set the password to 123, returned 0x00000000.
Success.
Selecting the key to be a non-migratable signing key of 2048 bits
Line76, main , Ask the TPM to create the requisite key - a non-
migratable signing key, not an AIK, returned 0x00000000. Success.
Line78, main , Assign policy to the key object, returned
0x00000000. Success.
Asking TPM to create a Non-migratable signing key. This may take
awhile
Line82, main , Create non-migratable signing key., returned
0x00000000. Success.
Loading the key into the TPM
Line91, main , Load the key into the TPM, returned 0x00000000.
Success.
Store the key in non volatile storage in the TPM, such that only
the owner can evict it
Line95, main , Make key an owner evict key, returned 0x00000000.
Success.
In order to not fll up the TPM with keys, change the key so it is
allowed to be evicted
Line100, main , Make the key evictabl, returned 0x00000000.
Success.
Storing public portion of key in MotherboardNMSK.pub
Line107, main , Get public portion of key, returned 0x00000000.
Success.
Finished writing fle
Save the encrypted key blob to MotherboardNMSK.blob
Line123, main , Get key blob, for removable media, returned
0x00000000. Success.
Finished writing fle
```

Рисунок 35 – Результат тестирования модуля удаления ключа

Результат тестирования модуля хеширования данных приведен на рисунке 36.

```
[root@localhost diplom]# ./tpm_gethash
(Line38, main) Create a context returned 0x00000000. Success.
(Line40, main) Connect to TPM returned 0x00000000. Success.
(Line44, main) Create Hash object returned 0x00000000. Success.
(Line48, main) Hash in the public key returned 0x00000000. Success.
(Line50, main) Get the hashed result returned 0x00000000. Success.
HASH: 5061626c6963206b657920746f20686173682e00
(Line65, main) Tspi Context Free Memory returned 0x00000000. Success.
(Line67, main) Tspi Context Close returned 0x00000000. Success.
```

Рисунок 36 – Результат тестирования модуля хеширования данных

Результат тестирования модуля подписи данных приведен на рисунке 37.

```
[root@localhost diplom]# ./tpm_signdata
(Line51, main) Create a context returned 0x00000000. Success.
(Line53, main) Connect to TPM returned 0x00000000. Success.
(Line55, main) Get TPM handle returned 0x00000000. Success.
(Line57, main) Get SRK handle returned 0x00000000. Success.
(Line59, main) Get SRK Policy returned 0x00000000. Success.
(Line61, main) Tspi_Policy_SetSecret returned 0x00000000. Success.
(Line67, main) Create policy object returned 0x00000000. Success.
(Line70, main) Set Policy secret returned 0x00000000. Success.
(Line77, main) Create the key object returned 0x00000000. Success.
(Line82, main) Set the keys's padding type returned 0x00000000. Success.
(Line84, main) Assign the key's policy to the key returned 0x00000000. Success.
Creating the key could take a while
(Line88, main) Asking TPM to create the key returned 0x00000000. Success.
(Line95, main) Registering the key for later retrieval returned 0x00000000. Success.
Registering the key blob for later retrieval
(Line100, main) Loading key in TPM returned 0x00000000. Success.
(Line103, main) Get Public portion of key returned 0x00000000. Success.
Finished writing SigningKey.pub
(Line114, main) Policy flush secret returned 0x00000000. Success.
(Line122, main) Get key by UUID returned 0x00000000. Success.
(Line124, main) Load private key returned 0x00000000. Success.
(Line129, main) Create Hash Object returned 0x00000000. Success.
(Line141, main) Hash the public key returned 0x00000000. Success.
(Line145, main) Sign returned 0x00000000. Success.
(Line153, main) Get key handle returned 0x00000000. Success.
Unregistering key
(Line159, main) Unregister key returned 0x00000000. Success.
(Line162, main) Tspi Context Free Memory returned 0x00000000. Success.
(Line164, main) Tspi Context Close returned 0x00000000. Success.
[root@localhost diplom]#
```

Рисунок 37 – Результат тестирования модуля подписи данных

Результат тестирования модуля подтверждения подлинности данных приведен на рисунке 38.

```
[root@localhost diplom]# ./tpm_verifysign
(Line51, main) Create a context returned 0x00000000. Success.
(Line53, main) Connect to TPM returned 0x00000000. Success.
(Line55, main) Get TPM handle returned 0x00000000. Success.
(Line57, main) Get SRK handle returned 0x00000000. Success.
(Line59, main) Get SRK Policy returned 0x00000000. Success.
(Line61, main) Tspi_Policy_SetSecret returned 0x00000000. Success.
(Line67, main) Create policy object returned 0x00000000. Success.
(Line70, main) Set Policy secret returned 0x00000000. Success.
(Line77, main) Create the key object returned 0x00000000. Success.
(Line82, main) Set the keys's padding type returned 0x00000000. Success.
(Line84, main) Assign the key's policy to the key returned 0x00000000.
Success.
Creating the key could take a while
(Line88, main) Asking TPM to create the key returned 0x00000000. Success.
(Line95, main) Registering the key for later retrieval returned
0x00000000. Success.
Registering the key blob for later retrieval
(Line100, main) Loading key in TPM returned 0x00000000. Success.
(Line103, main) Get Public portion of key returned 0x00000000. Success.
Finished writing SigningKey.pub
(Line114, main) Policy flush secret returned 0x00000000. Success.
(Line122, main) Get key by UUID returned 0x00000000. Success.
(Line124, main) Load private key returned 0x00000000. Success.
(Line129, main) Create Hash Object returned 0x00000000. Success.
(Line141, main) Hash the public key returned 0x00000000. Success.
(Line145, main) Sign returned 0x00000000. Success.
(Line162, main) Verify returned 0x00000000. Success.
(Line167, main) Tspi_GetPolicyObject returned 0x00000000. Success.
(Line172, main) Tspi_Policy_SetSecret returned 0x00000000. Success.
(Line178, main) Create MigObject returned 0x00000000. Success.
(Line183, main) Get Hash Value returned 0x00000000. Success.
(Line190, main) Set digest to migdata returned 0x00000000. Success.
(Line196, main) Set signature to migdata returned 0x00000000. Success.
(Line207, main) Create object verify key returned 0x00000000. Success.
(Line212, main) Set Pub Key in Verify Key returned 0x00000000. Success.
(Line214, main) Verify using CMKCreateTicket
Signature verified using TPM returned 0x00000000. Success.
(Line222, main) Get key handle returned 0x00000000. Success.
Unregistering key
(Line228, main) Unregister key returned 0x00000000. Success.
(Line231, main) Tspi Context Free Memory returned 0x00000000. Success.
(Line233, main) Tspi Context Close returned 0x00000000. Success.
```

Рисунок 38 – Результат тестирования модуля подтверждения подлинности данных

Результат тестирования модуля генерации случайных чисел приведен на рисунке 39.

```
[root@localhost diplom]# ./tpm_getrandom
(Line37, main) Create a context
 returned 0x00000000. Success.
(Line39, main) Connect to TPM
 returned 0x00000000. Success.
(Line41, main) Get TPM handle
 returned 0x00000000. Success.
(Line45, main) Get SRK handle
 returned 0x00000000. Success.
(Line48, main) Get SRK Policy
 returned 0x00000000. Success.
(Line51, main) Tspi_Policy_SetSecret
 returned 0x00000000. Success.
(Line54, main) Tspi_TPM_GetRandom
 returned 0x00000000. Success.
random:ee011b6224bcd7474212e6b8d1afb465
(Line62, main) Tspi Context Free Memory
 returned 0x00000000. Success.
(Line65, main) Tspi Context Close
 returned 0x00000000. Success.
```

Рисунок 39 – Результат тестирования модуля генерации случайных чисел

Результат тестирования модуля создания области энергонезависимой памяти NVRAM приведен на рисунке 40.

```
[root@localhost diplom]# ./tpm_createnvram
(Line29, main) Create a context returned 0x00000000. Success.
(Line31, main) Connect to TPM returned 0x00000000. Success.
(Line35, main) Create NV object returned 0x00000000. Success.
(Line38, main) Get TPM handle returned 0x00000000. Success.
(Line42, main) Set index returned 0x00000000. Success.
(Line45, main) Set policy returned 0x00000000. Success.
(Line48, main) Set size returned 0x00000000. Success.
(Line53, main) Get TPM Policy returned 0x00000000. Success.
(Line55, main) Set secret to 123 returned 0x00000000. Success.
(Line58, main) Created NV space returned 0x00000000. Success.
(Line61, main) Tspi Context Free Memory returned 0x00000000.
Success.
(Line63, main) Tspi Context Close returned 0x00000000. Success.
```

Рисунок 40 – Результат тестирования модуля создания области энергонезависимой памяти NVRAM

Результат тестирования модуля записи области энергонезависимой памяти NVRAM приведен на рисунке 41.

```
[root@localhost diplom]# ./tpm_writenvram
(Line29, main) Create a context returned 0x00000000. Success.
(Line31, main) Connect to TPM returned 0x00000000. Success.
(Line35, main) Create NV object returned 0x00000000. Success.
(Line38, main) Set index returned 0x00000000. Success.
(Line41, main) Set policy returned 0x00000000. Success.
(Line44, main) Set size returned 0x00000000. Success.
(Line49, main) Get TPM Policy returned 0x00000000. Success.
(Line51, main) Set secret to 123 returned 0x00000000. Success.
(Line53, main) Assigned policy to NV object returned 0x00000000. Success.
(Line56, main) Wrote to the NVRAM returned 0x00000000. Success.
(Line59, main) Tspi Context Free Memory returned 0x00000000. Success.
(Line61, main) Tspi Context Close returned 0x00000000. Success.
```

Рисунок 41 – Результат тестирования модуля записи области энергонезависимой памяти NVRAM

Результат тестирования модуля чтения области энергонезависимой памяти NVRAM приведен на рисунке 41.

```
[root@localhost diplom]# ./tpm_readnvram
(Line30, main) Create a context returned 0x00000000. Success.
(Line32, main) Connect to TPM returned 0x00000000. Success.
(Line36, main) Create NV object returned 0x00000000. Success.
(Line39, main) Set index returned 0x00000000. Success.
(Line42, main) Set policy returned 0x00000000. Success.
(Line45, main) Set size returned 0x00000000. Success.
(Line49, main) Read value from NV returned 0x00000000. Success.
Read data: й8(Line53, main) Tspi Context Free Memory returned 0x00000000. Success.
(Line55, main) Tspi Context Close returned 0x00000000. Success.
```

Рисунок 42 - Результат тестирования модуля чтения области энергонезависимой памяти NVRAM

Результат тестирования модуля удаления области энергонезависимой памяти NVRAM приведен на рисунке 43.

```
[root@localhost diplom]# ./tpm_destroynvram

(Line29, main) Create a context returned 0x00000000. Success.

(Line31, main) Connect to TPM returned 0x00000000. Success.

(Line35, main) Create an NVRAM object returned 0x00000000.
Success.

(Line39, main) Get TPM Object returned 0x00000000. Success.

(Line41, main) Get TPM Policy returned 0x00000000. Success.

(Line43, main) Set the TPM policy secret to 11111111 returned
0x00000000. Success.

(Line47, main) Set NVRAM index returned 0x00000000. Success.

Release NVRAM space

(Line52, main) Release NV space returned 0x00000000. Success.
```

Рисунок 43 - Результат тестирования модуля удаления области энергонезависимой памяти NVRAM

Результат тестирования модуля чтения регистров PCR приведен на рисунке 44.

```
[root@localhost diplom]# ./tpm_readpcr
(Line38, main) Create a context
 returned 0x00000000. Success.
(Line40, main) Connect to TPM
 returned 0x00000000. Success.
(Line42, main) Get TPM handle
 returned 0x00000000. Success.
(Line47, main) Get SRK handle
 returned 0x00000000. Success.
(Line50, main) Get SRK Policy
 returned 0x00000000. Success.
(Line52, main) Tspi_Policy_SetSecret
 returned 0x00000000. Success.

PCR List
PCR00: 0000000000000000000000000000000000000000
PCR01: 0000000000000000000000000000000000000000
PCR02: 0000000000000000000000000000000000000000
PCR03: 0000000000000000000000000000000000000000
PCR04: 0000000000000000000000000000000000000000
PCR05: 0000000000000000000000000000000000000000
PCR06: 0000000000000000000000000000000000000000
PCR07: 0000000000000000000000000000000000000000
PCR08: 0000000000000000000000000000000000000000
PCR09: 0000000000000000000000000000000000000000
PCR10: 0000000000000000000000000000000000000000
PCR11: 0000000000000000000000000000000000000000
PCR12: 0000000000000000000000000000000000000000
PCR13: 0000000000000000000000000000000000000000
PCR14: 0000000000000000000000000000000000000000
PCR15: 0000000000000000000000000000000000000000
PCR16: ffffffffffffffffffffffffffffffffffffffff
PCR17: ffffffffffffffffffffffffffffffffffffffff
PCR18: ffffffffffffffffffffffffffffffffffffffff
PCR19: ffffffffffffffffffffffffffffffffffffffff
PCR20: ffffffffffffffffffffffffffffffffffffffff
PCR21: ffffffffffffffffffffffffffffffffffffffff
PCR22: ffffffffffffffffffffffffffffffffffffffff
PCR23: ffffffffffffffffffffffffffffffffffffffff
```

Рисунок 44 – Результат тестирования модуля чтения регистров PCR

Результат тестирования модуля сброса регистров PCR приведен на рисунке 45.

```
[root@localhost diplom]# ./tpm_resetpcr
(Line34, main) Create a context returned 0x00000000. Success.
(Line36, main) Connect to TPM returned 0x00000000. Success.
(Line38, main) Get TPM handle returned 0x00000000. Success.
(Line41, main) Get Event returned 0x00000000. Success.
ulEventNumber:0
ulEventLength:0
rgbEvent:(null)
eventBlank
Event 0, is
eventBlank
Event 1, is
eventBlank
Event 2, is
eventBlank
Event 3, is
eventBlank
Event 4, is
eventBlank
Event 5, is
eventBlank
Event 6, is
eventBlank
Event 7, is
eventBlank
Event 8, is
eventBlank
Event 9, is
eventBlank
Event 10, is
eventBlank
Event 11, is
eventBlank
Event 12, is
eventBlank
Event 13, is
eventBlank
Event 14, is
(Line55, main) Tspi Context Free Memory returned 0x00000000. Success.
(Line57, main) Tspi Context Close returned 0x00000000. Success.
```

Рисунок 45 – Результат тестирования модуля сброса регистров PCR

Результат тестирования модуля смены регистров PCR приведен на рисунке 46.

```
[root@localhost diplom]# ./tpm_extendpcr
(Line44, main) Create a context
 returned 0x00000000. Success.
(Line46, main) Connect to TPM
 returned 0x00000000. Success.
(Line48, main) Get TPM handle
 returned 0x00000000. Success.

 Command line arguments:
 argv[0] ./tpm_extendpcr
(Line96, main) Tspi Context Free Memory
 returned 0x00000000. Success.
(Line98, main) Tspi Context Close
 returned 0x00000000. Success.
```

Рисунок 46 – Результат тестирования модуля смены регистров PCR

Результат тестирования модуля создания ключа подтверждения подлинности AIK приведен на рисунке 47.

```
[root@localhost diplom]# ./tpm_createaik
Line40, main , Create Context, returned 0x00000000. Success.
Line42, main , Context Connect, returned 0x00000000. Success.
Line45, main , Get TPM Handle, returned 0x00000000. Success.
Line48, main , Get the SRK handle, returned 0x00000000. Success.
Line52, main , GetTPM policy, returned 0x00000000. Success.
Line54, main , Set TPM policy's secret to 123, returned 0x00000000.
Success.
Line57, main , Create a policy object for the key, returned
0x00000000. Success.
Line60, main , Set the password to 123, returned 0x00000000.
Success.
Line66, main , Ask the TPM to create the requisite key - a non-
migratable signing key, not an AIK, returned 0x00000000. Success.
Line68, main , Assign policy to the key object, returned
0x00000000. Success.
Line71, main , Create Attestation Identity Keys, returned
0x00000000. Success.
Line74, main , Load the key into the TPM, returned 0x00000000.
Success.
Line78, main , Tspi Context Free Memory, returned 0x00000000.
Success.
Line80, main , Tspi Context Close, returned 0x00000000. Success.
```

Рисунок 47 – Результат тестирования модуля создания ключа подтверждения AIK

7 Вредные психофизические факторы, влияющие на нервную систему специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие

Человек, выполняя определенную работу, может сталкиваться с различными опасными ситуациями, которые способны нанести ему определенный вред.

Производственные факторы, в зависимости от сферы своего происхождения можно разделить на:

-факторы производственной среды, то есть пространства, где происходит трудовая деятельность;

-факторы трудового процесса.

При этом фактор трудового процесса основываясь на оценке возможности причинения вреду работника, подразделяют:

-неблагоприятные производственные;

-нейтральные и благоприятные производственные.

Неблагоприятные факторы, исходя из результирующего действия на организм делят на:

- вредные производственные, которые приводят и усугубляют болезни, имеющиеся у человека;

- опасные производственные, которые приводят к травмам, наносят физический вред.

Из совокупности данных факторов, базируясь на источнике происхождения, выделяют:

- психофизиологические;

- организационно-управленческие.

Нервно-психические перегрузки подразделяются на:

− умственное перенапряжение;

− перенапряжение анализаторов;

− монотонность труда. [16]

При выполнении трудовой деятельности, на организм специалиста по защите информации больше всего оказывают воздействия данные факторы. Они связаны с физическими и эмоциональными перегрузками, перенапряжением и длительным выполнением однообразных действий или непрерывной и устойчивой концентрации внимания в условиях дефицита сенсорных нагрузок.

Для снижения монотонности на человека нужно, во - первых, передислоцировать работающих с одной производственной операции на другую, а во-вторых, обеспечивать оптимальный режим труда и совокупности с грамотным распределением отдыха в течение рабочего дня. Человек может довольно долго выполнять умеренные умственные операции без отдыха, однако людям, который в основном заняты только им, необходим более длительный отдых.

Психофизический фактор определяется также обстановкой, внутри коллективными взаимоотношениями, уровнем обслуживания, соответствием работы способностям ее выполняющего. [17]

Таким образом, при выполнение данных методов нервной системе специалиста по защите информации будет наноситься минимальный вред.

Заключение

В выпускной квалификационной работе, которая выполнялась на предприятии АО ПНИЭИ были рассмотрены вопросы актуальности применения технологий доверенных вычислений, обзор технологий доверенных вычислений, таких как, технология Intel vPro, технология защиты данных на дисках Bitlocker, Next Generation Secure Computing и технология доверенных вычислений стандартов безопасности Trusted Computing Group. Были подробно рассмотрены и проанализированы приложения управления модулем TPM.

Были рассмотрены архитектура и основные возможности модуля Trusted Platform Module.

Также была произведена установка среды TPM, а именно эмулятора модуля, программного стека доверенных вычислений TrouSerS Trusted Group Software Stack и приложения TrouSerS TPM Tools.

В выпускной квалификационной работе было разработано программное средство тестирования и управления модулем доверенной платформы, алгоритм работы программного средства. Проведена отладка программного средства и тестирование модуля доверенной платформы, с использованием разработанного программного средства. В заключении, описаны вредные психофизические факторы, влияющие на нервную систему специалиста по защите информации, а также методы и средства физической культуры, снижающие их воздействие.

Таким образом, все задания на выпускную квалификационную работу были выполнены в полном объеме.

Список используемых источников

1 Технология Intel v-Pro [Электронный ресурс]. – Режим доступа: http://compress.ru/article.aspx?id=18057, свободный - Загл. с экрана;

2 Технология Intel v – Pro, Intel TXT [Электронный ресурс]. – Режим доступа: http://smb.ixbt.com/articles/tehnologii-i-produkty/2015-07-01/, свободный - Загл. с экрана;

3 Набор средств шифрования данных для ПК – анализ безопасности [Электронный ресурс]. – Режим доступа: https://technet.microsoft.com/ru-ru/library/cc162804.aspx, свободный - Загл. с экрана;

4 Основы безопасности для Microsoft следующего поколения – технические вопросы [Электронный ресурс]. – Режим доступа: https://msdn.microsoft.com/en-us/library/cc723472.aspx, свободный - Загл. с экрана;

5 Технология Palladium [Электронный ресурс]. – Режим доступа: https://www.itweek.ru/infrastructure/article/detail.php?ID=62612, свободный – Загл. с экрана;

6 Trusted Computing Group [Электронный ресурс]. – Режим доступа: https://trustedcomputinggroup.org/about, свободный – Загл. с экрана;

7 Trusted Platform Module [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Trusted_Platform_Module, свободный – Загл. с экрана;

8 Концепция Trusted Platform Module (TPM) [Электронный ресурс]. – Режим доступа: http://www.thg.ru/howto/200510082/print.html, свободный – Загл. с экрана;

9 Steven Kinney Trusted Platform Module Basics Using TPM in Embedded Systems;

10 Trusted Computing Group. TPM main specification. Main Specification Version 1.2 rev. 103, Trusted Computing Group, 2007;

11 Marcel Selhorst, Christian Stüble, Felix Teerkorn TSS Study Introduction and Analysis of the Open Source TCG Software Stack TrouSerS and Tools in its Environment;

12 СТО ПГУ 3.12 – 2015. ВЫПУСКНАЯ КВАЛИФИОННАЯ РАБОТА обучающихся по образовательным программам высшего образования – программ бакалавриата, программ специалиста и программ магистратуры. [Текст];

13 Anoosheh Zaerin The TPM Manager Requirements, Design, and Implementation Version 0.8 Sirrix AG security technolog Lise-Meitner-Allee 4 D-44801 Bochum, Germany, 2009 [Текст];

14 Trusted Computing Group. TCG software stack specification Version 1.2, Trusted Computing Group, 2006 [Текст]:

15 . Мэтью Н. Основы программирования в Linux: Пер. с англ. / Н. Мэтью, Р. Стоунс. – 4-е изд., перераб. И доп. – СПб.: БХВ-Петербург, 2009. – 896 с. [Текст];

16 ГОСТ 12.0.003-2015. Система стандартов по безопасности труда. Опасные и вредные производственные факторы. Классификация [Электронный ресурс] . — М.: Стандартинформ, 2016. - Режим доступа: http://www.internet-law.ru/gosts/gost/62075/. — Загл. с экрана;

17 Щедрина А. Г. Онтогенез и теория здоровья : методологические аспекты / А. Г. Щедрина. – Новосибирск : СО РАМН, 2003. – 164 с.

Приложение А

(обязательное)

Текст программных модулей тестирования и управления модулем доверенной платформы

Текст программы модуля создания ключа подписи TPM:

```
#define SIGN_KEY_UUID {0,0,0,0,0,{0,0,0,2,11}}
int main(int argc, char **argv)
{
    TSS_HCONTEXT hContexttpm=0;
    TSS_HTPM hTPMtpm = 0;
    TSS_RESULT resulttpm;
    TSS_HKEY hSRKtpm = 0;
    TSS_HPOLICY hSRKPolicytpm=0;
    TSS_UUID SRK_UUIDtpm = TSS_UUID_SRK;
    BYTE wkstpm[20];
    memset(wkstpm,0,20);

    TSS_HKEY hSigning_Keytpm;
    TSS_UUID MY_UUIDtpm=SIGN_KEY_UUIDtpm;
    TSS_HPOLICY hSigning_Key_Policytpm;
    TSS_FLAG initFlagstpm;
    BYTE *pubKeytpm;
    UINT32 pubKeySizetpm;
    FILE *fouttpm;

    resulttpm =Tspi_Context_Create(&hContexttpm);
    DBGTPM("Creat a cont", resulttpm);
    resulttpm=Tspi_Context_Connect(hContexttpm,
NULL);
    DBGTPM("Connect TPM", resulttpm);
    resulttpm=Tspi_Context_GetTpmObject(hContexttpm,
&hTPMtpm);
    DBGTPM("Get TPM handl", resulttpm);
    resulttpm=Tspi_Context_LoadKeyByUUID(hContexttpm
, TSS_PS_TYPE_SYSTEM, SRK_UUIDtpm, &hSRKtpm);
    DBGTPM("Get SRK handl", resulttpm);
    resulttpm=Tspi_GetPolicyObject(hSRKtpm,
TSS_POLICY_USAGE, &hSRKPolicytpm);
    DBGTPM("Get SRK Polisy", resulttpm);
    resulttpm=Tspi_Policy_SetSecret(hSRKPolicytpm,TS
S_SECRET_MODE_SHA1, 20, wkstpm);
    DBGTPM("Tspi_Policy_SetSecret", resulttpm);
```

```
        resulttpm                                                =
Tspi_Context_CreateObject(hContexttpm,
TSS_OBJECT_TYPE_POLICY,
                TSS_POLICY_USAGE,
&hSigning_Key_Policytpm);
        DBGTPM("Create policy object", resulttpm);
        resulttpm                                                =
Tspi_Policy_SetSecret(hSigning_Key_Policytpm,
                    TSS_SECRET_MODE_SHA1, 20, wkstpm);
        DBGTPM("Set Policy secret", resulttpm);
        initFlagstpm = TSS_KEY_TYPE_SIGNING |
                TSS_KEY_SIZE_2048 |
                TSS_KEY_NO_AUTHORIZATION |
                TSS_KEY_NOT_MIGRATABLE;
        resulttpm                                                =
Tspi_Context_CreateObject(hContexttpm,
TSS_OBJECT_TYPE_RSAKEY,
                initFlagstpm, &hSigning_Keytpm);
        DBGTPM("Create the key object", resulttpm);
        resulttpm = Tspi_SetAttribUint32(hSigning_Keytpm,
            TSS_TSPATTRIB_KEY_INFO,
            TSS_TSPATTRIB_KEYINFO_ENCSCHEME,
            TSS_ES_RSAESPKCSV15);
        DBGTPM("Set the key pading type", resulttpm);
        resulttpm                                                =
Tspi_Policy_AssignToObject(hSigning_Key_Policytpm,hSignin
g_Keytpm);
        DBGTPM("Assign the key polisy to key", resulttpm);
        printf("Creat the key could take a while\n");
        resulttpm = Tspi_Key_CreateKey(hSigning_Keytpm,
            hSRKtpm, 0);
        DBGTPM("Ask TPM to creat the key", resulttpm);
        resulttpm = Tspi_Context_RegisterKey(hContexttpm,
                        hSigning_Keytpm,
                        TSS_PS_TYPE_SYSTEM,
                        MY_UUIDtpm,
                        TSS_PS_TYPE_SYSTEM,
                        SRK_UUIDtpm);
```

```
        DBGTPM("Register the key for later retreval",
resulttpm);

        printf("Register the key blob for later
retreval\r\n");

        resulttpm                                    =
Tspi_Key_LoadKey(hSigning_Keytpm,hSRKtpm);
        DBGTPM("Load key in TPM", resulttpm);
        resulttpm = Tspi_Key_GetPubKey(hSigning_Keytpm,
                    &pubKeySizetpm, &pubKeytpm);
        DBGTPM("Get Publ portion of key", resulttpm);
        fouttpm = fopen("SigningKey.pub", "wb");
        if(fouttpm != NULL) {
            write(fileno(fouttpm),            pubKeytpm,
pubKeySizetpm);
            printf("Finishd writing SigningKey.pub\n");
            fclose(fouttpm);
        }
        else {
            printf("Error opening XXXXXXXXXXXX \r\n");
        }
        resulttpm                                    =
Tspi_Policy_FlushSecret(hSigning_Key_Policytpm);
        DBGTPM("Policy flush secret", resulttpm);

        resulttpm                                    =
Tspi_Context_GetKeyByUUID(hContexttpm,
                          TSS_PS_TYPE_SYSTEM,
                          MY_UUIDtpm,
                          &hSigning_Keytpm);
        DBGTPM("Get key handle", resulttpm);
        printf("Unregistering key\r\n");
        resulttpm                                    =
Tspi_Context_UnregisterKey(hContexttpm,
                          TSS_PS_TYPE_SYSTEM,
                          MY_UUIDtpm,
                          &hSigning_Keytpm);
        DBGTPM("Unregistr key", resulttpm);
```

```
        resulttpm = Tspi_Context_FreeMemory(hContexttpm,
NULL);
        DBGTPM("Tispi Context free memory", resulttpm);
        resulttpm = Tspi_Context_Close(hContexttpm);
        DBGTPM("Tispi Context Close", resulttpm);
        return 0;
    }
```

Текст программы модуля создания ключа привязки в TPM:

```
    int main(int argc, char **argv)
    {
        TSS_HCONTEXT hContexttpm=0;
        TSS_HTPM hTPMtpm = 0;
        TSS_RESULT resulttpm;
        TSS_HKEY hSRKtpm = 0;
        TSS_HPOLICY hSRKPolicytpm=0;
        TSS_UUID SRK_UUIDtpm = TSS_UUID_SRK;
        BYTE wkstpm[20];
        memset(wkstpm,0,20);

        TSS_HKEY hESS_Bind_Keytpm;
        TSS_UUID ESS_BIND_UUIDtpm=BACKUP_KEY_UUIDtpm;
        TSS_HPOLICY hESS_Policytpm;
        TSS_FLAG initFlagstpm;
        BYTE *pubKeytpm;
        UINT32 pubKeySizetpm;
        FILE *fouttpm;

        resulttpm =Tspi_Context_Create(&hContexttpm);
        DBGTPM("Create a context", resulttpm);
```

```
        resulttpm=Tspi_Context_Connect(hContexttpm,
NULL);
        DBGTPM("Connect to TPM", resulttpm);
        resulttpm=Tspi_Context_GetTpmObject(hContexttpm,
&hTPMtpm);
        DBGTPM("Get TPM handle", resulttpm);
        resulttpm=Tspi_Context_LoadKeyByUUID(hContexttpm
, TSS_PS_TYPE_SYSTEM, SRK_UUIDtpm, &hSRKtpm);
        DBGTPM("Get SRK handle", resulttpm);
        resulttpm=Tspi_GetPolicyObject(hSRKtpm,
TSS_POLICY_USAGE, &hSRKPolicytpm);
        DBGTPM("Get SRK Policy", resulttpm);
        resulttpm=Tspi_Policy_SetSecret(hSRKPolicytpm,TS
S_SECRET_MODE_SHA1, 20, wkstpm);
        DBGTPM("Tspi_Policy_SetSecret", resulttpm);
        resulttpm                                    =
Tspi_Context_CreateObject(hContexttpm,TSS_OBJECT_TYPE_POL
ICY,
                TSS_POLICY_USAGE, &hESS_Policytpm);
        DBGTPM("Create a key polisy object", resulttpm);
        resulttpm = Tspi_Policy_SetSecret(hESS_Policytpm,
TSS_SECRET_MODE_SHA1,
                20, wkstpm);
        DBGTPM("Set key polisy object secret", resulttpm);
        initFlagstpm = TSS_KEY_TYPE_BIND |
                TSS_KEY_SIZE_2048 |
                TSS_KEY_AUTHORIZATION |
                TSS_KEY_NOT_MIGRATABLE;
```

```
        resulttpm =
Tspi_Context_CreateObject(hContexttpm,
TSS_OBJECT_TYPE_RSAKEY,
                initFlagstpm, &hESS_Bind_Keytpm);
        DBGTPM("Create the key object", resulttpm);
        resulttpm =
Tspi_SetAttribUint32(hESS_Bind_Keytpm,
            TSS_TSPATTRIB_KEY_INFO,
            TSS_TSPATTRIB_KEYINFO_ENCSCHEME,
            TSS_ES_RSAESPKCSV15);
        DBGTPM("Set the key padding type", resulttpm);
        resulttpm =
Tspi_Policy_AssignToObject(hESS_Policytpm,
hESS_Bind_Keytpm);
        DBGTPM("Assign the key polisy the key",
resulttpm);
        printf("Creat the key could take a while\n");
        resulttpm = Tspi_Key_CreateKey(hESS_Bind_Keytpm,
            hSRKtpm, 0);
        DBGTPM("Ask TPM to create the key", resulttpm);
        resulttpm = Tspi_Context_RegisterKey(hContexttpm,
                        hESS_Bind_Keytpm,
                        TSS_PS_TYPE_SYSTEM,
                        ESS_BIND_UUIDtpm,
                        TSS_PS_TYPE_SYSTEM,
                        SRK_UUIDtpm);
        DBGTPM("Register the key for later retreval",
resulttpm);
        printf("Register the key blob for later
retreval\r\n");
```

```
        resulttpm                                          =
Tspi_Key_LoadKey(hESS_Bind_Keytpm,hSRKtpm);
        DBGTPM("Load key in TPM", resulttpm);
        resulttpm = Tspi_Key_GetPubKey(hESS_Bind_Keytpm,
                    &pubKeySizetpm, &pubKeytpm);
        DBGTPM("get Public portion of key", resulttpm);
        fouttpm = fopen("BackupESSBindKey.pub", "wb");
        if(fouttpm != NULL) {
            write(fileno(fouttpm),              pubKeytpm,
pubKeySizetpm);
            printf("Finish                              write
BackupESSBindKey.pub\n");
            fclose(fouttpm);
        }
        else {
            printf("Eror opening XXXXXXXXXXXX \r\n");
        }
        resulttpm                                          =
Tspi_Policy_FlushSecret(hESS_Policytpm);
        DBGTPM("Policy flush secret", resulttpm);

        resulttpm  =  Tspi_Context_FreeMemory(hContexttpm,
NULL);
        DBGTPM("Tispi Context Free Memory", resulttpm);
        resulttpm = Tspi_Context_Close(hContexttpm);
        DBGTPM("Tispi Context Close", resulttpm);
        return 0;
    }
```

Текст программы модуля шифрования данных:

```
int main(int argc, char **argv)
```

```
    {
        TSS_HCONTEXT hContexttpm=0;
        TSS_HTPM hTPMtpm = 0;
        TSS_RESULT resulttpm;
        TSS_HKEY hSRKtpm = 0;
        TSS_HPOLICY hSRKPolicytpm=0;
        TSS_UUID SRK_UUIDtpm = TSS_UUID_SRK;
        BYTE wkstpm[20];
        memset(wkstpm,0,20);

        TSS_HKEY hESS_Bind_Keytpm;
        TSS_UUID ESS_BIND_UUIDtpm=BACKUP_KEY_UUIDtpm;
        TSS_HPOLICY hESS_Policytpm;
        TSS_FLAG initFlagstpm;
        BYTE *pubKeytpm;
        UINT32 pubKeySizetpm;
        FILE *fouttpm, *fintpm;

        resulttpm =Tspi_Context_Create(&hContexttpm);
        DBGTPM("Create a context", resulttpm);
        resulttpm=Tspi_Context_Connect(hContexttpm,
NULL);
        DBGTPM("Connect to TPM", resulttpm);
        resulttpm=Tspi_Context_GetTpmObject(hContexttpm,
&hTPMtpm);
        DBGTPM("Get TPM handle", resulttpm);
        resulttpm=Tspi_Context_LoadKeyByUUID(hContexttpm
, TSS_PS_TYPE_SYSTEM, SRK_UUIDtpm, &hSRKtpm);
        DBGTPM("Get SRK handle", resulttpm);
```

```
        resulttpm=Tspi_GetPolicyObject(hSRKtpm,
TSS_POLICY_USAGE, &hSRKPolicytpm);
        DBGTPM("Get SRK Policy", resulttpm);
        resulttpm=Tspi_Policy_SetSecret(hSRKPolicytpm,TS
S_SECRET_MODE_SHA1, 20, wkstpm);
        DBGTPM("Tispi_Policy_SetSecret", resulttpm);
        resulttpm                                       =
Tspi_Context_CreateObject(hContexttpm,TSS_OBJECT_TYPE_POL
ICY,
                TSS_POLICY_USAGE, &hESS_Policytpm);
        DBGTPM("Create a key polisy object", resulttpm);
        resulttpm = Tspi_Policy_SetSecret(hESS_Policytpm,
TSS_SECRET_MODE_SHA1,
                20, wkstpm);
        DBGTPM("Set key policy object secret", resulttpm);
        initFlagstpm = TSS_KEY_TYPE_BIND |
                TSS_KEY_SIZE_2048 |
                TSS_KEY_AUTHORIZATION |
                TSS_KEY_NOT_MIGRATABLE;
        resulttpm                                       =
Tspi_Context_CreateObject(hContexttpm,
TSS_OBJECT_TYPE_RSAKEY,
                initFlagstpm, &hESS_Bind_Keytpm);
        DBGTPM("Create key object", resulttpm);
        resulttpm                                       =
Tspi_SetAttribUint32(hESS_Bind_Keytpm,
            TSS_TSPATTRIB_KEY_INFO,
            TSS_TSPATTRIB_KEYINFO_ENCSCHEME,
            TSS_ES_RSAESPKCSV15);
        DBGTPM("Set the key pading type", resulttpm);
```

```
        resulttpm                                          =
Tspi_Policy_AssignToObject(hESS_Policytpm,
hESS_Bind_Keytpm);
        DBGTPM("Assign the key polisy to key", resulttpm);
        printf("Creating the key could take a while\n");
        resulttpm = Tspi_Key_CreateKey(hESS_Bind_Keytpm,
            hSRKtpm, 0);
        DBGTPM("Ask TPM to create the key", resulttpm);
        resulttpm = Tspi_Context_RegisterKey(hContexttpm,
                        hESS_Bind_Keytpm,
                        TSS_PS_TYPE_SYSTEM,
                        ESS_BIND_UUIDtpm,
                        TSS_PS_TYPE_SYSTEM,
                        SRK_UUIDtpm);
        DBGTPM("Register the key for later retreval",
resulttpm);

        printf("Register the key blob for later
retrieval\r\n");

        resulttpm                                          =
Tspi_Key_LoadKey(hESS_Bind_Keytpm,hSRKtpm);
        DBGTPM("Loading a key in TPM", resulttpm);
        resulttpm = Tspi_Key_GetPubKey(hESS_Bind_Keytpm,
                    &pubKeySizetpm, &pubKeytpm);
        DBGTPM("Get Public portion of key", resulttpm);
        fouttpm = fopen("BackupESSBindKey.pub", "wb");
        if(fouttpm != NULL) {
            write(fileno(fouttpm),              pubKeytpm,
pubKeySizetpm);
```

```
            printf("Finishd                          writing
BackupESSBindKey.pub\n");
            fclose(fouttpm);
        }
        else {
            printf("Eror opening XXXXXXXXXXXX \r\n");
        }
        resulttpm                                        =
Tspi_Policy_FlushSecret(hESS_Policytpm);
        DBGTPM("Policy flush secret", resulttpm);

        resulttpm                                        =
Tspi_Context_GetKeyByUUID(hContexttpm,
                          TSS_PS_TYPE_SYSTEM,
                          ESS_BIND_UUIDtpm,
                          &hESS_Bind_Keytpm);
        DBGTPM("Get key handle", resulttpm);
        printf("Unregister key\r\n");
        resulttpm                                        =
Tspi_Context_UnregisterKey(hContexttpm,
                          TSS_PS_TYPE_SYSTEM,
                          ESS_BIND_UUIDtpm,
                          &hESS_Bind_Keytpm);
        DBGTPM("Unregister key", resulttpm);

        UINT32 ulDataLength;
        BYTE *rbgBoundData;
        BYTE newPubKeytpm[1000], encData[1000];
        TSS_HENCDATA hEncData;
        fintpm = fopen("BackupESSBindKey.pub", "r");
```

```
        read(fileno(fintpm), newPubKeytpm,284);
        if (fintpm == NULL)
            return 0;
        fclose(fintpm);
        resulttpm                                          =
Tspi_Context_CreateObject(hContexttpm,
                    TSS_OBJECT_TYPE_RSAKEY,
                    initFlagstpm, &hESS_Bind_Keytpm);
        DBGTPM("Tispi_Context_CreateObject      BindKey",
resulttpm);
        resulttpm = Tspi_SetAttribData(hESS_Bind_Keytpm,
                    TSS_TSPATTRIB_KEY_BLOB,
                    TSS_TSPATTRIB_KEYBLOB_PUBLIC_KEY,
                    284,newPubKeytpm);
        DBGTPM("Set  Public  key  into  new  key  object",
resulttpm);
        fintpm = fopen("data", "rb");
        read(fileno(fintpm), encData, 7);
        fclose(fintpm);
        resulttpm=Tspi_Context_CreateObject(hContexttpm,
                    TSS_OBJECT_TYPE_ENCDATA,
                    TSS_ENCDATA_BIND,
                    &hEncData);
        DBGTPM("Create gata Object", resulttpm);
        resulttpm                                          =
Tspi_Data_Bind(hEncData,hESS_Bind_Keytpm,7,encData);
        DBGTPM("Bind data", resulttpm);
        resulttpm = Tspi_GetAttribData(hEncData,
                TSS_TSPATTRIB_ENCDATA_BLOB,
                TSS_TSPATTRIB_ENCDATABLOB_BLOB,
```

```
                &ulDataLength,&rbgBoundData);
        DBGTPM("Get encryptd data", resulttpm);
        fouttpm = fopen("Bound.data", "wb");
        write(fileno(fouttpm),rbgBoundData,ulDataLength)
;
        fclose(fouttpm);
        resulttpm  =  Tspi_Context_FreeMemory(hContexttpm,
NULL);
        DBGTPM("Tispi Context Free Memory", resulttpm);
        resulttpm = Tspi_Context_Close(hContexttpm);
        DBGTPM("Tspi Context Close", resulttpm);
        return 0;
    }
```

Текст программы модуля расшифрования данных:

```
    int main(int argc, char **argv)
    {

        TSS_HCONTEXT hContexttpm=0;
        TSS_HTPM hTPMtpm = 0;
        TSS_RESULT resulttpm;
        TSS_HKEY hSRKtpm = 0;
        TSS_HPOLICY hSRKPolicytpm=0;
        TSS_UUID SRK_UUIDtpm = TSS_UUID_SRK;
        BYTE wkstpm[20];
        memset(wkstpm,0,20);

        TSS_HKEY hESS_Bind_Keytpm;
        TSS_UUID ESS_BIND_UUIDtpm=BACKUP_KEY_UUIDtpm;
        TSS_HPOLICY hESS_Policytpm;
        TSS_FLAG initFlagstpm;
        BYTE *pubKeytpm;
        UINT32 pubKeySizetpm;
        FILE *fouttpm, *fintpm;
        resulttpm =Tspi_Context_Create(&hContexttpm);
        DBGTPM("Create a context", resulttpm);
        resulttpm=Tspi_Context_Connect(hContexttpm,
NULL);
        DBGTPM("Connect to TPM", resulttpm);
```

```
        resulttpm=Tspi_Context_GetTpmObject(hContexttpm,
&hTPMtpm);
        DBGTPM("Get TPM handle", resulttpm);
        resulttpm=Tspi_Context_LoadKeyByUUID(hContexttpm
, TSS_PS_TYPE_SYSTEM, SRK_UUIDtpm, &hSRKtpm);
        DBGTPM("Get SRK handle", resulttpm);
        resulttpm=Tspi_GetPolicyObject(hSRKtpm,
TSS_POLICY_USAGE, &hSRKPolicytpm);
        DBGTPM("Get SRK Polisy", resulttpm);
        resulttpm=Tspi_Policy_SetSecret(hSRKPolicytpm,TS
S_SECRET_MODE_SHA1, 20, wkstpm);
        DBGTPM("Tispi_Policy_SetSecret", resulttpm);


        resulttpm                                       =
Tspi_Context_CreateObject(hContexttpm,TSS_OBJECT_TYPE_POL
ICY,
                TSS_POLICY_USAGE, &hESS_Policytpm);
        DBGTPM("Create a key policy object", resulttpm);
        resulttpm = Tspi_Policy_SetSecret(hESS_Policytpm,
TSS_SECRET_MODE_SHA1,
                20, wkstpm);
        DBGTPM("Set   key   policy   object   secrets",
resulttpm);
        initFlagstpm = TSS_KEY_TYPE_BIND |
                TSS_KEY_SIZE_2048 |
                TSS_KEY_AUTHORIZATION |
                TSS_KEY_NOT_MIGRATABLE;
        resulttpm                                       =
Tspi_Context_CreateObject(hContexttpm,
TSS_OBJECT_TYPE_RSAKEY,
                initFlagstpm, &hESS_Bind_Keytpm);
        DBGTPM("Create the key object", resulttpm);
        resulttpm                                       =
Tspi_SetAttribUint32(hESS_Bind_Keytpm,
            TSS_TSPATTRIB_KEY_INFO,
            TSS_TSPATTRIB_KEYINFO_ENCSCHEME,
            TSS_ES_RSAESPKCSV15);
        DBGTPM("Set the key pading type", resulttpm);
        resulttpm                                       =
Tspi_Policy_AssignToObject(hESS_Policy,
hESS_Bind_Keytpm);
        DBGTPM("Assign   the   key   policy   the   key",
resulttpm);
        printf("Creat the key could take a while\n");
```

```
        resulttpm = Tspi_Key_CreateKey(hESS_Bind_Keytpm,
            hSRKtpm, 0);
        DBGTPM("Ask TPM to create the key", resulttpm);
        resulttpm = Tspi_Context_RegisterKey(hContexttpm,
                        hESS_Bind_Keytpm,
                        TSS_PS_TYPE_SYSTEM,
                        ESS_BIND_UUIDtpm,
                        TSS_PS_TYPE_SYSTEM,
                        SRK_UUIDtpm);
        DBGTPM("Register the key for later retreval",
resulttpm);

        printf("Registering the key blob for later
retreval\r\n");

        resulttpm                                    =
Tspi_Key_LoadKey(hESS_Bind_Keytpm,hSRKtpm);
        DBGTPM("Loading key in TPM", resulttpm);
        resulttpm = Tspi_Key_GetPubKey(hESS_Bind_Keytpm,
                     &pubKeySizetpm, &pubKeytpm);
        DBGTPM("Get Public portion of key", resulttpm);
        fouttpm = fopen("BackupESSBindKey.pub", "wb");
        if(fouttpm != NULL) {
            write(fileno(fouttpm),            pubKeytpm,
pubKeySizetpm);
            printf("Finishd                       writing
BackupESSBindKey.pub\n");
            fclose(fouttpm);
        }
        else {
            printf("Eror opening XXXXXXXXXXXX \r\n");
        }
        resulttpm                                    =
Tspi_Policy_FlushSecret(hESS_Policytpm);
        DBGTPM("Policy flush secret", resulttpm);

        UINT32 ulDataLengthtpm;
        BYTE *rbgBoundDatatpm;
        BYTE newPubKeytpm[1000], encDatatpm[1000];
        TSS_HENCDATA hEncDatatpm;
        fintpm = fopen("BackupESSBindKey.pub", "r");
        read(fileno(fintpm), newPubKeytpm,284);
        if (fintpm == NULL)
            return 0;
        fclose(fintpm);
```

```
        resulttpm                                              =
Tspi_Context_CreateObject(hContexttpmtpm,
                    TSS_OBJECT_TYPE_RSAKEY,
                    initFlagstpm, &hESS_Bind_Keytpmtpm);
        DBGTPM("Tspi_Context_CreateObject       BindKey",
resulttpm);
        resulttpm = Tspi_SetAttribData(hESS_Bind_Keytpm,
                    TSS_TSPATTRIB_KEY_BLOB,
                    TSS_TSPATTRIB_KEYBLOB_PUBLIC_KEY,
                    284,newPubKeytpm);
        DBGTPM("Set  Public  key  into  new  key  object",
resulttpm);
        fintpm = fopen("data", "rb");
        read(fileno(fintpm), encDatatpm, 7);
        fclose(fintpm);
        resulttpm=Tspi_Context_CreateObject(hContexttpm,
                    TSS_OBJECT_TYPE_ENCDATA,
                    TSS_ENCDATA_BIND,
                    &hEncDatatpmtpm);
        DBGTPM("Create Data Object", resulttpm);
        resulttpm                                              =
Tspi_Data_Bind(hEncDatatpm,hESS_Bind_Keytpm,7,encDatatpm)
;
        DBGTPM("Bind data", resulttpm);
        resulttpm = Tspi_GetAttribData(hEncDatatpm,
                TSS_TSPATTRIB_ENCDATA_BLOB,
                TSS_TSPATTRIB_ENCDATABLOB_BLOB,
                &ulDataLengthtpm,&rbgBoundDatatpm);
        DBGTPM("Get encrypted data", resulttpm);
        fouttpm = fopen("Bound.data", "wb");
        write(fileno(fouttpm),rbgBoundDatatpm,ulDataLeng
thtpm);
        fclose(fouttpm);

        UINT32 encLentpm=256;
        BYTE encryptedDatatpm[256];
        BYTE *rgbDataunBoundtpm;
        TSS_HKEY                              hUnBind_Keytpm,
hRecovered_UnBind_Keytpm;

        fintpm = fopentpm("Bound.data", "rb");
        read(fileno(fintpm),                 encryptedDatatpm,
encLentpm);
        fclose(fintpm);
```

```
        resulttpm                                              =
Tspi_Context_CreateObject(hContexttpm,
                          TSS_OBJECT_TYPE_ENCDATA,
                          TSS_ENCDATA_BIND, &hEncDatatpm);
        DBGTPM("Created Data Object", resulttpm);
        resulttpm = Tspi_SetAttribData(hEncDatatpm,
                          TSS_TSPATTRIB_ENCDATA_BLOB,
                          TSS_TSPATTRIB_ENCDATABLOB_BLOB,
                          encLentpm, encryptedDatatpm);
        DBGTPM("Set Attrib data to hEncdata", resulttpm);
        resulttpm                                              =
Tspi_Context_GetKeyByUUID(hContexttpm,
                          TSS_PS_TYPE_SYSTEM,
                          ESS_BIND_UUIDtpm,
                          &hUnBind_Keytpm);
        DBGTPM("Get unbinding key", resulttpm);
        resulttpm = Tspi_Key_LoadKey(hUnBind_Keytpm,
                         hSRKtpm);
        DBGTPM("Loaded Key", resulttpm);
        resulttpm = Tspi_Data_Unbind(hEncDatatpm,
                         hUnBind_Keytpm,
                         &ulDataLengthtpm,
                         &rgbDataunBoundtpm);
        DBGTPM("Unbind", resulttpm);
        fouttpm = fopen("output", "wb");
        write(fileno(fouttpm),         rgbDataunBoundtpm,
ulDataLengthtpm);
        fclose(fouttpm);

        resulttpm                                              =
Tspi_Context_GetKeyByUUID(hContexttpm,
                          TSS_PS_TYPE_SYSTEM,
                          ESS_BIND_UUIDtpm,
                          &hESS_Bind_Keytpm);
        DBGTPM("Get key handle", resulttpm);
        printf("Unregistering key\r\n");
        resulttpm                                              =
Tspi_Context_UnregisterKey(hContexttpm,
                          TSS_PS_TYPE_SYSTEM,
                          ESS_BIND_UUIDtpm,
                          &hESS_Bind_Keytpm);
        DBGTPM("Unregister key", resulttpm);

        resulttpm  =  Tspi_Context_FreeMemory(hContexttpm,
NULL);
```

```
        DBGTPM("Tispi Context Free Memory", resulttpm);
        resulttpm = Tspi_Context_Close(hContexttpm);
        DBGTPM("Tsp Context Close", resulttpm);
        return 0;
    }
```

Текст программы модуля опечатывания данных:

```
    int main(int argc, char **argv)
    {
        TSS_HCONTEXT hContexttpm=0;
        TSS_HTPM hTPMtpm = 0;
        TSS_RESULT resulttpm;
        TSS_HKEY hSRKtpm = 0;
        TSS_HPOLICY hSRKPolicytpm=0;
        TSS_UUID SRK_UUIDtpm = TSS_UUID_SRK;

        TSS_HPCRS hPCRstpm = 0;
        BYTE TypePasstpm[12]="My Password";
        UINT32 ulPcrLentpm;
        BYTE *rbgPcrValuetpm;
        UINT32 encDataSizetpm;
        BYTE *encDatatpm;
        FILE *fouttpm;
        TSS_HENCDATA hEncDatatpm = 0;
        //SRKtpm - 20 bytes 0
        BYTE wkstpm[20];
        memset(wkstpm,0,20);

        resulttpm =Tspi_Context_Create(&hContexttpm);
        DBGTPM("Create a context", resulttpm);
        resulttpm=Tspi_Context_Connect(hContexttpm,
NULL);
        DBGTPM("Connect to TPM", resulttpm);
        resulttpm=Tspi_Context_GetTpmObject(hContexttpm,
&hTPMtpm);
        DBGTPM("Get TPM handle", resulttpm);

        resulttpm=Tspi_Context_LoadKeyByUUID(hContexttpm
, TSS_PS_TYPE_SYSTEM, SRK_UUIDtpm, &hSRKtpm);
        DBGTPM("Get SRK handle", resulttpm);
        resulttpm=Tspi_GetPolicyObject(hSRKtpm,
TSS_POLICY_USAGE, &hSRKPolicytpm);
        DBGTPM("Get SRK Policy", resulttpm);
        resulttpm=Tspi_Policy_SetSecret(hSRKPolicytpm,TS
S_SECRET_MODE_SHA1,20, wkstpm);
```

```
        DBGTPM("Tspi_Policy_SetSecret", resulttpm);

        //Seal Data
        resulttpm                                          =
Tspi_Context_CreateObject(hContexttpm,
                          TSS_OBJECT_TYPE_PCRS,
                          0,
                          &hPCRstpm);
        DBGTPM("Createing a PCR object", resulttpm);
        resulttpm = Tspi_TPM_PcrRead(hTPMtpm, 8,
                    &ulPcrLentpm, &rbgPcrValuetpm);
        DBGTPM("Read the current value for PcR 8",
resulttpm);
        resulttpm                                          =
Tspi_PcrComposite_SetPcrValue(hPCRstpm,8,20,rbgPcrValuetp
m);
        DBGTPM("Set the value read from PCR_8 as seal
value of PCR_8", resulttpm);

        resulttpm = Tspi_TPM_PcrRead(hTPMtpm, 9,
                    &ulPcrLentpm, &rbgPcrValuetpm);
        DBGTPM("Read the current value for PCR 9",
resulttpm);
        resulttpm                                          =
Tspi_PcrComposite_SetPcrValue(hPCRstpm,9,20,rbgPcrValuetp
m);
        DBGTPM("Set the value read from PCR_9 as sealing
value of PCR_9", resulttpm);

        resulttpm                                          =
Tspi_Context_CreateObject(hContexttpm,
                          TSS_OBJECT_TYPE_ENCDATA,
                          TSS_ENCDATA_SEAL,
                          &hEncDatatpm);
        DBGTPM("Create data object for sealing",
resulttpm);
        resulttpm = Tspi_Data_Seal(hEncDatatpm,
                    hSRKtpm,
                    strlen((char*)TypePasstpm),
                    TypePasstpm,hPCRstpm);
        DBGTPM("Sealed my password, using the SRK key, to
PCR 9", resulttpm);
        //write in file
        resulttpm = Tspi_GetAttribData(hEncDatatpm,
                    TSS_TSPATTRIB_ENCDATA_BLOB,
```

```
                    TSS_TSPATTRIB_ENCDATABLOB_BLOB,
                    &encDataSizetpm,
                    &encDatatpm);
        DBGTPM("Get the encrypted seal data", resulttpm);
        fouttpm = fopen("owner_auth.pass", "wb");
        if(fouttpm != NULL)
        {
            write(fileno(fouttpm),          encDatatpm,
encDataSizetpm);
            fclose(fouttpm);
        }
        else {
            printf("Error opening Owner_Auth.pass!");
        }

        resulttpm  =  Tspi_Context_FreeMemory(hContexttpm,
NULL);
        DBGTPM("Tspi Context Free Memory", resulttpm);
        resulttpm = Tspi_Context_Close(hContexttpm);
        DBGTPM("Tspi Context Close", resulttpm);
        return 0;
    }
```

Текст программы модуля распечатывания данных:

```
int main(int argc, char **argv)
{
    TSS_HCONTEXT hContexttpm=0;
    TSS_HTPM hTPMtpm = 0;
    TSS_RESULT resulttpm;
    TSS_HKEY hSRKtpm = 0;
    TSS_HPOLICY hSRKPolicytpm=0;
    TSS_UUID SRK_UUIDtpm = TSS_UUID_SRK;
    //SRKtpm - 20bytes 0

    BYTE wkstpm[20];
    memset(wkstpm,0,20);
    resulttpm =Tspi_Context_Create(&hContexttpm);
    DBGTPM("Create a context\n", resulttpm);
```

```
        resulttpm=Tspi_Context_Connect(hContexttpm,
NULL);
        DBGTPM("Conect to TPM\n", resulttpm);
        resulttpm=Tspi_Context_GetTpmObject(hContexttpm,
&hTPMtpm);
        DBGTPM("Get TPM handle\n", resulttpm);

        resulttpm=Tspi_Context_LoadKeyByUUID(hContexttpm
, TSS_PS_TYPE_SYSTEM, SRK_UUIDtpm, &hSRKtpm);
        DBGTPM("Get SRK handle\n", resulttpm);
        resulttpm=Tspi_GetPolicyObject(hSRKtpm,
TSS_POLICY_USAGE, &hSRKPolicytpm);
        DBGTPM("Get SRK Policy\n", resulttpm);
        resulttpm=Tspi_Policy_SetSecret(hSRKPolicytpm,TS
S_SECRET_MODE_SHA1,20, wkstpm);
        DBGTPM("Tspi_Policy_SetSecret\n", resulttpm);
        //Unseal Data
        UINT32 outlengthtpm;
        BYTE   *outstringtpm;
        BYTEEncryptedDatatpm[312];
        FILE   *fintpm;
        TSS_HENCDATA hRetrieveDatatpm = 0;
        memset(EncryptedDatatpm, 0, 312);
        fintpm = fopen("owner_auth.pass", "rb");
        read(fileno(fintpm), EncryptedDatatpm,312);
        fclose(fintpm);
        resulttpm                                    =
Tspi_Context_CreateObject(hContexttpm,
                    TSS_OBJECT_TYPE_ENCDATA,
                    TSS_ENCDATA_SEAL,
```

```
                    &hRetrieveDatatpm);
        DBGTPM("Create   data   object   for   unseal",
resulttpm);
        resulttpm = Tspi_SetAttribData(hRetrieveDatatpm,
                    TSS_TSPATTRIB_ENCDATA_BLOB,
                    TSS_TSPATTRIB_ENCDATABLOB_BLOB,
                    312,EncryptedDatatpm);
        DBGTPM("Set  attribut  data  to  hRetrieveData",
resulttpm);
        resulttpm                                       =
Tspi_Data_Unseal(hRetrieveDatatpm,hSRKtpm,&outlengthtpm,&
outstringtpm);
        DBGTPM("Unseal data!", resulttpm);
        outstringtpm[outlengthtpm]=0;
        printf("%d, %s\n",outlengthtpm, outstringtpm);

        resulttpm  =  Tspi_Context_FreeMemory(hContexttpm,
NULL);
        DBGTPM("Tispi Context Free Memory", resulttpm);
        resulttpm = Tspi_Context_Close(hContexttpm);
        DBGTPM("Tspi Context Close", resulttpm);
        return 0;
    }
```

Текст программы удаления ключа:

```
    int main(int argc,char **argv)
    {
        TSS_HCONTEXT hContexttpm   = 0;
        TSS_HTPM  hTPMtpm    = 0;
        TSS_RESULT    resulttpm;
        TSS_HKEY  hSRKtpm       = 0;
```

```
        TSS_HPOLICY  hTPM_Policytpm = 0;
        BYTE     wkstpm[2];

        TSS_HKEY hMB_AIK_Keytpm=0;
        TSS_FLAG initFlagstpm;
        TSS_HPOLICY hMB_AIK_Policytpm;
        UINT32 blob_lengthtpm;
        BYTE *mb_aik_key_blobtpm;
        BYTE *pubKeytpm;
        UINT32 pubKeySizetpm;
        FILE *fouttpm;
        TSS_UUID pUuidDatatpm;
        TSS_UUID SRK_UUIDtpm   = TSS_UUID_SRK;

        memset(wkstpm,0,20);// Set wks to the well known
secret: 20 bytes of all 0's
        printf("Entering CreateOEAIK Temporary\r\n");
    resulttpm = Tspi_Context_Create(&hContexttpm);
        DBGTPM("Create Context", resulttpm);
        resulttpm   =   Tspi_Context_Connect(hContexttpm,
NULL);
        DBGTPM("Context Connect", resulttpm);
        resulttpm                                          =
Tspi_Context_GetTpmObject(hContexttpm, &hTPMtpm);
        DBGTPM("Get TPM Handle", resulttpm);
        resulttpm                                          =
Tspi_Context_LoadKeyByUUID(hContexttpm,
TSS_PS_TYPE_SYSTEM, SRK_UUIDtpm, &hSRKtpm);
        DBGTPM("Get the SRK handle", resulttpm);
```

```
        resulttpm=Tspi_GetPolicyObject(hTPMtpm,
TSS_POLICY_USAGE, &hTPM_Policytpm);
        DBGTPM("GetTPM policy",resulttpm);
        resulttpm=Tspi_Policy_SetSecret(hTPM_Policytpm,
TSS_SECRET_MODE_PLAIN, 3, (BYTE *)"123");
        DBGTPM("Set TPM policy secret to 123", resulttpm);

        resulttpm= Tspi_Context_CreateObject(hContexttpm,
TSS_OBJECT_TYPE_POLICY,  TSS_POLICY_USAGE,
&hMB_AIK_Policytpm);
        DBGTPM("Create  a  policy  object  for  the
key",resulttpm);
        resulttpm=Tspi_Policy_SetSecret(hMB_AIK_Policytp
m, TSS_SECRET_MODE_PLAIN, 3, (BYTE *)"123");
        DBGTPM("Set the password 123", resulttpm);
        printf("Select the key to be a non-migratable sign
key of 2048 bit\r\n");
        initFlagstpm     =     TSS_KEY_SIZE_2048     |
TSS_OBJECT_TYPE_RSAKEY |
                    TSS_KEY_TYPE_SIGNING               |
TSS_KEY_AUTHORIZATION | TSS_KEY_NOT_MIGRATABLE;
        resulttpm=Tspi_Context_CreateObject( hContexttpm,
TSS_OBJECT_TYPE_RSAKEY, initFlagstpm, &hMB_AIK_Keytpm );
        DBGTPM("Ask the TPM to create the requisite key -
a non-migratable sign key, not a AIK", resulttpm);
        resulttpm=Tspi_Policy_AssignToObject(
hMB_AIK_Policytpm, hMB_AIK_Keytpm);
        DBGTPM("Assign  policy  to  the  key  object",
resulttpm);
```

```
        printf("Ask TPM to create a Non-migratable sign
key. This may take awhile\n");
        resulttpm=Tspi_Key_CreateKey(hMB_AIK_Keytpm,
hSRKtpm, 0);
        DBGTPM("Create non-migratable signing key.",
resulttpm);
        printf("Load the key into the TPM\r\n");
        resulttpm=Tspi_Key_LoadKey(hMB_AIK_Keytpm,
hSRKtpm);
        DBGTPM("Load the key into the TPM", resulttpm);
        printf("Store the key in no volatile storage in
the TPM, such that onlyy the owner can evict it\n");
        resulttpm=Tspi_TPM_KeyControlOwner(hTPMtpm,
hMB_AIK_Keytpm,      TSS_TSPATTRIB_KEYCONTROL_OWNEREVICT,
TRUE, &pUuidDatatpm);
        DBGTPM("Make key an owner evict key", resulttpm);
        printf("In order to not full up the TPM with keys,
change the key so it is allowed to be evict\n");
        resulttpm=Tspi_TPM_KeyControlOwner(hTPMtpm,
hMB_AIK_Keytpm,      TSS_TSPATTRIB_KEYCONTROL_OWNEREVICT,
FALSE, &pUuidDatatpm);
        DBGTPM("Make the key evictabl", resulttpm);
        printf("Storing public portion of key in
MotherboardNMSK.pub\r\n");
        resulttpm=Tspi_Key_GetPubKey(hMB_AIK_Keytpm,
&pubKeySizetpm, &pubKeytpm);
        DBGTPM("Get public portion of key", resulttpm);
        fflush(stdouttpm);
        fouttpm=fopen( "MotherboardNMSK.pub", "wb");
        if(fouttpm != NULL){
```

```
            write(fileno(fouttpm),
pubKeytpm,pubKeySizetpm);
        fclose(fouttpm);
        printf("Finished writing fle\n");
        }
        else {
            printf("ERROR openig fle for writin\r\n");
        }
        printf("Save the encrypt key blob to
MotherboardNMSK.blob\n");
        resulttpm=Tspi_GetAttribData(hMB_AIK_Keytpm,
TSS_TSPATTRIB_KEY_BLOB,
                                TSS_TSPATTRIB_KEYBLOB_BLOB,
&blob_lengthtpm, &mb_aik_key_blobtpm);
        DBGTPM("Get key blob, for removable media",
resulttpm);
        fouttpm=fopen( "MotherboardNMSK.blob", "wb");
        if(fouttpm != NULL)
        {
            write(fileno(fouttpm),   mb_aik_key_blobtpm,
blob_lengthtpm);
            fclose(fouttpm);
            printf("Finished writing fle\n");
        }
        else {
            printf("ERROR opening fle for writing\r\n");
        }
        Tspi_Context_Close(hContexttpm);
        Tspi_Context_FreeMemory(hContexttpm, NULL);
        Tspi_Context_Close(hContexttpm);
```

```
        return 0;}
```

Текст программы модуля хеширования данных

```
int main(int argc, char **argv)
{
    TSS_HCONTEXT hContexttpm = 0;
    TSS_RESULT resulttpm;
    BYTE *digesttpm;
    UINT32 digestLentpm;
    TSS_HHASH hHashOfESSKeytpm;
    BYTE initialHashtpm[20];
    BYTE *pubKeytpm = (BYTE *)("Pablic key to hash.");//
public key to hash
    UINT32 pubKeysizetpm = 20;
    BYTE hashtpm[20];
    memset(initialHashtpm, 0, 20);
    UINT32 i;
    BYTE wkstpm[20];
    memset(wkstpm, 0, 20);
    resulttpm = Tspi_Context_Create(&hContexttpm);
    DBGTPM("Create a context", resulttpm);
    resulttpm = Tspi_Context_Connect(hContexttpm, NULL);
    DBGTPM("Connect to TPM", resulttpm);
    resulttpm  =  Tspi_Context_CreateObject(hContexttpm,
TSS_OBJECT_TYPE_HASH, TSS_HASH_SHA1, &hHashOfESSKeytpm);
    DBGTPM("Create Hash object", resulttpm);
    resulttpm  =  Tspi_Hash_SetHashValue(hHashOfESSKeytpm,
pubKeysizetpm, pubKeytpm);
    DBGTPM("Hash in the pablic key", resulttpm);
```

```
    resulttpm = Tspi_Hash_GetHashValue(hHashOfESSKeytpm,
&digestLentpm, &digesttpm);
    DBGTPM("Get the hashed result", resulttpm);
    if (resulttpm != TSS_SUCCESS)
    {
        resulttpm = Tspi_Context_FreeMemory(hContexttpm,
digesttpm);
        DBGTPM("Tispi Context Free Memory", resulttpm);
    }
    memcpy(hashtpm, digesttpm, 20);
    printf("HASH: ");
    for (i = 0; i<20; i++)
        printf("%02x", hashtpm[i]);
    printf("\n");
    resulttpm   =   Tspi_Context_FreeMemory(hContexttpm,
NULL);
    DBGTPM("Tspi Context Free Memory", resulttpm);
    resulttpm = Tspi_Context_Close(hContexttpm);
    DBGTPM("Tspi Context Close", resulttpm);
    return 0;
}
```

Текст программы модуля подписи данных:

```
    UINT32 filelengthtpm(char *filenametpm)
    {
        struct stat st;
        if (filename != NULL)
        {
            stat(filenametpm, &st);
            return st.st_size;
        }
    }

    int main(int argc, char **argv)
```

```
    {

        TSS_HCONTEXT hContexttpm = 0;
        TSS_HTPM hTPMtpm = 0;
        TSS_RESULT resulttpm;
        TSS_HKEY hSRKtpm = 0;
        TSS_HPOLICY hSRKPolicytpm = 0;
        TSS_UUID SRK_UUID = TSS_UUID_SRK;
        BYTE wkstpm[20];
        memset(wkstpm, 0, 20);

        TSS_HKEY hSigning_Keytpm;
        TSS_UUID MY_UUID = SIGN_KEY_UUID;
        TSS_HPOLICY hSigning_Key_Policytpm;
        TSS_FLAG initFlagstpm;
        BYTE *pubKeytpm;
        UINT32 pubKeySizetpm;
        FILE *fouttpm, *fintpm;

        resulttpm = Tspi_Context_Create(&hContexttpm);
        DBG("Create a context", resulttpm);
        resulttpm   =   Tspi_Context_Connect(hContexttpm,
NULL);
        DBG("Conect to TPM", resulttpm);
        resulttpm                                          =
Tspi_Context_GetTpmObject(hContexttpm, &hTPMtpmtpm);
        DBG("Get TPM handle", resulttpm);
        resulttpm                                          =
Tspi_Context_LoadKeyByUUID(hContexttpm,
TSS_PS_TYPE_SYSTEM, SRK_UUID, &hSRKtpm);
        DBG("Get SRK handle", resulttpm);
        resulttpm     =     Tspi_GetPolicyObject(hSRKtpm,
TSS_POLICY_USAGE, &hSRKPolicytpm);
        DBG("Get SRK Policy", resulttpm);
        resulttpm  =  Tspi_Policy_SetSecret(hSRKPolicytpm,
TSS_SECRET_MODE_SHA1, 20, wkstpm);
        DBG("Tspi_Policy_SetSecret", resulttpm);
```

```
        resulttpm                                              =
Tspi_Context_CreateObject(hContexttpm,
TSS_OBJECT_TYPE_POLICY,
            TSS_POLICY_USAGE, &hSigning_Key_Policytpm);
        DBG("Create policy object", resulttpm);
        resulttpm                                              =
Tspi_Policy_SetSecret(hSigning_Key_Policytpm,
            TSS_SECRET_MODE_SHA1, 20, wkstpm);
        DBG("Set Policy secret", resulttpm);
        initFlagstpm = TSS_KEY_TYPE_SIGNING |
            TSS_KEY_SIZE_2048 |
            TSS_KEY_NO_AUTHORIZATION |
            TSS_KEY_NOT_MIGRATABLE;
        resulttpm                                              =
Tspi_Context_CreateObject(hContexttpm,
TSS_OBJECT_TYPE_RSAKEY,
            initFlagstpm, &hSigning_Keytpm);
        DBG("Creat the key object", resulttpm);
        resulttpm = Tspi_SetAttribUint32(hSigning_Keytpm,
            TSS_TSPATTRIB_KEY_INFO,
            TSS_TSPATTRIB_KEYINFO_ENCSCHEME,
            TSS_ES_RSAESPKCSV15);
        DBG("Set the keys's padding type", resulttpm);
        resulttpm                                              =
Tspi_Policy_AssignToObject(hSigning_Key_Policytpm,
hSigning_Keytpm);
        DBG("Assign the key's policy to the key",
resulttpm);
        printf("Creating the key could take a while\n");
        resulttpm = Tspi_Key_CreateKey(hSigning_Keytpm,
            hSRKtpm, 0);
        DBG("Ask TPM to create the key", resulttpm);
        resulttpm = Tspi_Context_RegisterKey(hContexttpm,
            hSigning_Keytpm,
            TSS_PS_TYPE_SYSTEM,
            MY_UUID,
            TSS_PS_TYPE_SYSTEM,
            SRK_UUID);
```

```
        DBG("Registerig the key for later retrieval",
resulttpm);

        printf("Registerig the key blob for later
retreval\r\n");

        resulttpm = Tspi_Key_LoadKey(hSigning_Keytpm,
hSRKtpm);
        DBG("Load key in TPM", resulttpm);
        resulttpm = Tspi_Key_GetPubKey(hSigning_Keytpm,
            &pubKeySizetpm, &pubKeytpm);
        DBG("Get Public portion of key", resulttpm);
        fouttpm = fopen("SigningKey.pub", "wb");
        if (fouttpm != NULL) {
            write(fileno(fouttpm),          pubKeytpm,
pubKeySizetpm);
            printf("Finishd writing SigningKey.pub\n");
            fclose(fouttpm);
        }
        else {
            printf("Eror opening XXXXXXXXXXXX \r\n");
        }
        resulttpm                                       =
Tspi_Policy_FlushSecret(hSigning_Key_Policytpm);
        DBG("Policy flush secret", resulttpm);

        TSS_HHASH hHashToSign = 0;
        resulttpm                                       =
Tspi_Context_GetKeyByUUID(hContexttpm,
            TSS_PS_TYPE_SYSTEM,
            MY_UUID,
            &hSigning_Keytpm);
        DBG("Get key by UUID", resulttpm);
        resulttpm = Tspi_Key_LoadKey(hSigning_Keytpm,
hSRKtpm);
        DBG("Load private key", resulttpm);
        resulttpm                                       =
Tspi_Context_CreateObject(hContexttpm,
            TSS_OBJECT_TYPE_HASH,
```

```
            TSS_HASH_SHA1,
            &hHashToSign);
        DBG("Create Hash Object", resulttpm);
        UINT32 pubKeyLength;
        BYTE pPubKey[284];
        UINT32 ulSignatureLength;
        BYTE *rgbSignature;
        pubKeyLength = filelength("SigningKey.pub");
        fintpm = fopen("SigningKey.pub", "rb");
        read(fileno(fintpm), pPubKey, pubKeyLength);
        fclose(fintpm);
        resulttpm                                          =
Tspi_Hash_UpdateHashValue(hHashToSign,
            pubKeyLength, pPubKey);
        DBG("Hash the public key", resulttpm);
        resulttpm      =      Tspi_Hash_Sign(hHashToSign,
hSigning_Keytpm,
            &ulSignatureLength,
            &rgbSignature);
        DBG("Sign", resulttpm);
        resulttpm                                          =
Tspi_Context_GetKeyByUUID(hContexttpm,
            TSS_PS_TYPE_SYSTEM,
            MY_UUID,
            &hSigning_Keytpm);
        DBG("Get key handle", resulttpm);
        printf("Unregistering key\r\n");
        resulttpm                                          =
Tspi_Context_UnregisterKey(hContexttpm,
            TSS_PS_TYPE_SYSTEM,
            MY_UUID,
            &hSigning_Keytpm);
        DBG("Unregistr key", resulttpm);

        resulttpm  =  Tspi_Context_FreeMemory(hContexttpm,
NULL);
        DBG("Tspi Context Free Memory", resulttpm);
        resulttpm = Tspi_Context_Close(hContexttpm);
        DBG("Tspi Context Close", resulttpm);
```

```
    return 0;
}
```

Текст программы модуля подтверждения подлинности данных:

```
UINT32 filelengthtpm(char *filenametpm)
{
    struct stat st;
    if(filenametpm != NULL)
    {
        stat(filenametpm, &st);
        return st.st_size;
    }
    return 0;
}

int main(int argc, char **argv)
{
    TSS_HCONTEXT hContexttpm=0;
    TSS_HTPM hTPMtpm = 0;
    TSS_RESULT resulttpm;
    TSS_HKEY hSRKtpm = 0;
    TSS_HPOLICY hSRKPolicytpm = 0, hTPM_Policytpm = 0;
    TSS_UUID SRK_UUIDtpm = TSS_UUID_SRK;
    BYTE wkstpm[20];
    memset(wkstpm,0,20);

    TSS_HKEY hSigning_Keytpm;
    TSS_UUID MY_UUIDtpm=SIGN_KEY_UUIDtpm;
    TSS_HPOLICY hSigning_Key_Policytpm;
    TSS_FLAG initFlagstpm;
    BYTE *pubKeytpm;
    UINT32 pubKeySizetpm;
```

```
        FILE *fouttpm, *fintpm;
        resulttpm =Tspi_Context_Create(&hContexttpm);
        DBGTPM("Creat a context", resulttpm);
        resulttpm=Tspi_Context_Connect(hContexttpm,
NULL);
        DBGTPM("Conect to TPM", resulttpm);
        resulttpm=Tspi_Context_GetTpmObject(hContexttpm,
&hTPMtpm);
        DBGTPM("Get TPM handle", resulttpm);
        resulttpm=Tspi_Context_LoadKeyByUUID(hContexttpm
, TSS_PS_TYPE_SYSTEM, SRK_UUIDtpm, &hSRKtpm);
        DBGTPM("Get SRK handle", resulttpm);
        resulttpm=Tspi_GetPolicyObject(hSRKtpm,
TSS_POLICY_USAGE, &hSRKPolicytpm);
        DBGTPM("Get SRK Policy", resulttpm);
        resulttpm=Tspi_Policy_SetSecret(hSRKPolicytpm,TS
S_SECRET_MODE_SHA1, 20, wkstpm);
        DBGTPM("Tspi_Policy_SetSecret", resulttpm);
        resulttpm                                       =
Tspi_Context_CreateObject(hContexttpm,
TSS_OBJECT_TYPE_POLICY,
                TSS_POLICY_USAGE,
&hSigning_Key_Policytpm);
        DBGTPM("Create policy object", resulttpm);
        resulttpm                                       =
Tspi_Policy_SetSecret(hSigning_Key_Policytpm,
                    TSS_SECRET_MODE_SHA1, 20, wkstpm);
        DBGTPM("Set Policy secret", resulttpm);
        initFlagstpm = TSS_KEY_TYPE_SIGNING |
                TSS_KEY_SIZE_2048 |
```

```
                TSS_KEY_NO_AUTHORIZATION |
                TSS_KEY_NOT_MIGRATABLE;
        resulttpm                                          =
Tspi_Context_CreateObject(hContexttpm,
TSS_OBJECT_TYPE_RSAKEY,
                initFlagstpm, &hSigning_Keytpm);
        DBGTPM("Creat the key object", resulttpm);
        resulttpm = Tspi_SetAttribUint32(hSigning_Keytpm,
            TSS_TSPATTRIB_KEY_INFO,
            TSS_TSPATTRIB_KEYINFO_ENCSCHEME,
            TSS_ES_RSAESPKCSV15);
        DBGTPM("Set the keys's padding type", resulttpm);
        resulttpm                                          =
Tspi_Policy_AssignToObject(hSigning_Key_Policytpm,hSignin
g_Keytpm);
        DBGTPM("Assign the key policy to the key",
resulttpm);
        printf("Creat the key could take a while\n");
        resulttpm = Tspi_Key_CreateKey(hSigning_Keytpm,
            hSRKtpm, 0);
        DBGTPM("Ask TPM to create the key", resulttpm);
        resulttpm = Tspi_Context_RegisterKey(hContexttpm,
                        hSigning_Keytpm,
                        TSS_PS_TYPE_SYSTEM,
                        MY_UUIDtpm,
                        TSS_PS_TYPE_SYSTEM,
                        SRK_UUIDtpm);
        DBGTPM("Registerng the key for later retreval",
resulttpm);
```

```
        printf("Registering the key blob for later
retreval\r\n");
        resulttpm =
Tspi_Key_LoadKey(hSigning_Keytpm,hSRKtpm);
        DBGTPM("Loadeng key in TPM", resulttpm);
        resulttpm = Tspi_Key_GetPubKey(hSigning_Keytpm,
                    &pubKeySizetpm, &pubKeytpm);
        DBGTPM("Get Public portion of key", resulttpm);
        fouttpm = fopen("SigningKey.pub", "wb");
        if(fouttpm != NULL) {
            write(fileno(fouttpm),            pubKeytpm,
pubKeySizetpm);
            printf("Finishd writing SigningKey.pub\n");
            fclose(fouttpm);
        }
        else {
            printf("Error opening XXXXXXXXXXXX \r\n");
        }
        resulttpm =
Tspi_Policy_FlushSecret(hSigning_Key_Policytpm);
        DBGTPM("Policy flush secret", resulttpm);
        TSS_HHASH hHashToSigntpm = 0;
        resulttpm =
Tspi_Context_GetKeyByUUID(hContexttpm,
                          TSS_PS_TYPE_SYSTEM,
                          MY_UUIDtpm,
                          &hSigning_Keytpm);
        DBGTPM("Get key by UUID", resulttpm);
        resulttpm =
Tspi_Key_LoadKey(hSigning_Keytpm,hSRKtpm);
```

```
        DBGTPM("Load private key", resulttpm);
        resulttpm                                          =
Tspi_Context_CreateObject(hContexttpm,
                    TSS_OBJECT_TYPE_HASH,
                    TSS_HASH_SHA1,
                    &hHashToSigntpm);
        DBGTPM("Create Hash Object", resulttpm);
        UINT32 pubKeyLengthtpm;
        BYTE pPubKeytpm[284];
        UINT32 ulSignatureLengthtpm;
        BYTE *rgbSignaturetpm;
        pubKeyLengthtpm                                    =
filelengthtpm("SigningKey.pub");
        fintpm = fopen("SigningKey.pub", "rb");
        read(fileno(fintpm),                    pPubKeytpm,
pubKeyLengthtpm);
        fclose(fintpm);
        resulttpm                                          =
Tspi_Hash_UpdateHashValue(hHashToSigntpm,
                        pubKeyLengthtpm, pPubKeytpm);
        DBGTPM("Hash the public key", resulttpm);
        resulttpm                                          =
Tspi_Hash_Sign(hHashToSigntpm,hSigning_Keytpm,
                    &ulSignatureLengthtpm,
                    &rgbSignaturetpm);
        DBGTPM("Sign", resulttpm);
        fouttpm = fopen("Signature.data", "wb");
        write(fileno(fouttpm),rgbSignaturetpm,ulSignatur
eLengthtpm);
        fclose(fouttpm);
```

```
        //Verify signature
        BYTE Signaturetpm[256];
        fintpm = fopen("Signature.data", "rb");
        read(fileno(fintpm), Signaturetpm, 256);
        resulttpm                                          =
Tspi_Hash_VerifySignature(hHashToSigntpm,hSigning_Keytpm,
256,Signaturetpm);
        DBGTPM("Verify", resulttpm);
        //Using TPM to verify the signature
        //Get TPM Policy Object
        resulttpm                                          =
Tspi_GetPolicyObject(hTPMtpm,TSS_POLICY_USAGE,&hTPM_Polic
ytpm);
        DBGTPM("Tspi_GetPolicyObject", resulttpm);
        //set secret use owner secret
        resulttpm = Tspi_Policy_SetSecret(hTPM_Policytpm,
                    TSS_SECRET_MODE_PLAIN,
                    6, (BYTE *)"123456");
        DBGTPM("Tspi_Policy_SetSecret", resulttpm);
        TSS_HMIGDATA hMigDatatpm = 0;
        resulttpm                                          =
Tspi_Context_CreateObject(hContexttpm,
                    TSS_OBJECT_TYPE_MIGDATA,
                    0,&hMigDatatpm);
        DBGTPM("Create MigObject", resulttpm);
        //Get Hash value
        UINT32 hashSizetpm;
        BYTE *myHashtpm;
```

```
        resulttpm                                          =
Tspi_Hash_GetHashValue(hHashToSigntpm,&hashSizetpm,&myHas
htpm);
        DBGTPM("Get Hash Value", resulttpm);
        //Set Attrib Data
        resulttpm = Tspi_SetAttribData(hMigDatatpm,
                    TSS_MIGATTRIB_TICKET_DATA,
                    TSS_MIGATTRIB_TICKET_SIG_DIGEST,
                    20,
                    myHashtpm);
        DBGTPM("Set digest to migdata", resulttpm);
        resulttpm = Tspi_SetAttribData(hMigDatatpm,
                    TSS_MIGATTRIB_TICKET_DATA,
                    TSS_MIGATTRIB_TICKET_SIG_VALUE,
                    256,
                    Signaturetpm);
        DBGTPM("Set signature to migdata", resulttpm);
        //Creat veryfying key
        TSS_HKEY hVerify_Keytpm;
        initFlagstpm = TSS_KEY_TYPE_SIGNING |
                TSS_KEY_SIZE_2048 |
                TSS_KEY_NO_AUTHORIZATION |
                TSS_KEY_MIGRATABLE;
        resulttpm                                          =
Tspi_Context_CreateObject(hContexttpm,
                          TSS_OBJECT_TYPE_RSAKEY,
                          initFlagstpm,
                          &hVerify_Keytpm);
        DBGTPM("Create object verify key", resulttpm);
        resulttpm = Tspi_SetAttribData(hVerify_Keytpm,
```

```
                        TSS_TSPATTRIB_KEY_BLOB,
                        TSS_TSPATTRIB_KEYBLOB_PUBLIC_KEY,
                        pubKeyLength,pPubKeytpm);
        DBGTPM("Set Pub Key in Verify Key", resulttpm);
        resulttpm   =   Tspi_TPM_CMKCreateTicket(hTPMtpm,
hVerify_Keytpm, hMigDatatpm);
        DBGTPM("Verify  using  CMKCreateTicket\nSignature
verified using TPM", resulttpm);
        //Unregistr the key if necesary
        resulttpm                                          =
Tspi_Context_GetKeyByUUID(hContexttpm,
                            TSS_PS_TYPE_SYSTEM,
                            MY_UUIDtpm,
                            &hSigning_Keytpm);
        DBGTPM("Get key handle", resulttpm);
        printf("Unregistering key\r\n");
        resulttpm                                          =
Tspi_Context_UnregisterKey(hContexttpm,
                            TSS_PS_TYPE_SYSTEM,
                            MY_UUIDtpm,
                            &hSigning_Keytpm);
        DBGTPM("Unregister key", resulttpm);
        resulttpm=   Tspi_Context_FreeMemory(hContexttpm,
NULL);
        DBGTPM("Tispi Context Free Memory", resulttpm);
        resulttpm = Tspi_Context_Close(hContexttpm);
        DBGTPM("Tspi Context Close", resulttpm);
        return 0;
    }
```

Текст программы модуля генерации случайных чисел:

```
int main(int argc, char **argv)
{
    TSS_HCONTEXT hContexttpm = 0;
    TSS_HTPM     hTPMtpm     = 0;
    TSS_RESULT   resulttpm;
    TSS_HKEY     hSRKtpm     = 0;
    TSS_HPOLICY  hSRKPolicytpm = 0;
    TSS_UUID     SRK_UUIDtpm   = TSS_UUID_SRK;
    UINT32       i, ulRandomDataLengthtpm = 16;
    BYTE*       prgbRandomDatatpm = 0;
    BYTE wkstpm[20]; //well known secret
    memset(wkstpm,0,20);
    resulttpm = Tspi_Context_Create(&hContexttpm);
    DBGTPM("Creat a context\n", resulttpm);
    resulttpm   =   Tspi_Context_Connect(hContexttpm,
NULL);
    DBGTPM("Conect to TPM\n", resulttpm);
    resulttpm                                          =
Tspi_Context_GetTpmObject(hContexttpm, &hTPMtpm);
    DBGTPM("Get TPM handle\n", resulttpm);
    resulttpm                                          =
Tspi_Context_LoadKeyByUUID(hContexttpm,
TSS_PS_TYPE_SYSTEM, SRK_UUIDtpm, &hSRKtpm);
    DBGTPM("Get SRK handle\n", resulttpm);
    resulttpm    =    Tspi_GetPolicyObject(hSRKtpm,
TSS_POLICY_USAGE, &hSRKPolicytpm);
    DBGTPM("Get SRK Policy\n", resulttpm);
    // Set the SRK policy to bi the wel knoun secret
```

```
        resulttpm = Tspi_Policy_SetSecret(hSRKPolicytpm,
TSS_SECRET_MODE_SHA1, 20, wkstpm);
        DBGTPM("Tspi_Policy_SetSecret\n", resulttpm);
        resulttpm      =      Tspi_TPM_GetRandom(hTPMtpm,
ulRandomDataLengthtpm, &prgbRandomDatatpm);
        DBGTPM("Tspi_TPM_GetRandom\n", resulttpm);
        printf("random:");
        for(i=0;i<ulRandomDataLengthtpm;i++)
            printf("%02x", prgbRandomDatatpm[i]);
        printf("\n");
        resulttpm = Tspi_Context_FreeMemory(hContexttpm,
NULL);
        DBGTPM("Tspi Context Free Memory\n", resulttpm);
        resulttpm = Tspi_Context_Close(hContexttpm);
        DBGTPM("Tspi Context Close\n", resulttpm);
        return 0;
    }
```

Текст программы модуля создания NVRAM:

```
    int main(int argc, char **argv) {
        TSS_HCONTEXT hContexttpm = 0;
        TSS_HNVSTORE hNVStoretpm = 0;
        TSS_HPOLICY  hTPMPolicytpm = 0;
        TSS_HTPM     hTPMtpm = 0;
        TSS_RESULT   resulttpm;
        resulttpm =Tspi_Context_Create(&hContexttpm);
        DBGTPM("Creat a context", resulttpm);
        resulttpm=Tspi_Context_Connect(hContexttpm,
NULL);
        DBGTPM("Connect to TPM", resulttpm);
        //Creat a NVRAM
```

```
        resulttpm=Tspi_Context_CreateObject(hContexttpm,
TSS_OBJECT_TYPE_NV, 0, &hNVStoretpm);
        DBGTPM("Create NV object", resulttpm);
        resulttpm                                          =
Tspi_Context_GetTpmObject(hContexttpm, &hTPMtpm);
        DBGTPM("Get TPM handle", resulttpm);
        resulttpm=Tspi_SetAttribUint32(hNVStoretpm,
TSS_TSPATTRIB_NV_INDEX, 0,0x00011101);
        DBGTPM("Set index",resulttpm);
        resulttpm=Tspi_SetAttribUint32(hNVStoretpm,
TSS_TSPATTRIB_NV_PERMISSIONS, 0, 0x2000);
        DBGTPM("Set polics",resulttpm);
        // 40 bytes
        resulttpm=Tspi_SetAttribUint32(hNVStoretpm,
TSS_TSPATTRIB_NV_DATASIZE, 0,40);
        DBGTPM("Set size",resulttpm);
        resulttpm=Tspi_GetPolicyObject(hTPMtpm,
TSS_POLICY_USAGE, &hTPMPolicytpm);
        DBGTPM("Get TPM Policy",resulttpm);
        // Set secret
        resulttpm=   Tspi_Policy_SetSecret(hTPMPolicytpm,
TSS_SECRET_MODE_PLAIN, 3, (BYTE *)"123");
        DBGTPM("Set secret to 123",resulttpm);
        // Creat the NVRAM space
        resulttpm=Tspi_NV_DefineSpace(hNVStoretpm,0,0);
        DBGTPM("Creat NV space",resulttpm);
        resulttpm  =  Tspi_Context_FreeMemory(hContexttpm,
NULL);
        DBGTPM("Tispi Context Free Memory", resulttpm);
        resulttpm = Tspi_Context_Close(hContexttpm);
```

```
        DBGTPM("Tspi Context Close", resulttpm);
        return 0;
    }
```

Текст программы модуля записи в область NVRAM:

```
int main(int argc, char **argv)
{
    TSS_HCONTEXT hContexttpm = 0;
    TSS_HNVSTORE hNVStoretpm = 0;
    TSS_RESULT   resulttpm;
    TSS_HPOLICY hNewPolicytpm;
    char dataToStoretpm[6]="Data.";
    resulttpm =Tspi_Context_Create(&hContexttpm);
    DBGTPM("Create a context", resulttpm);
    resulttpm=Tspi_Context_Connect(hContexttpm, NULL);
    DBGTPM("Connect to TPM", resulttpm);
    resulttpm=Tspi_Context_CreateObject(hContexttpm,
TSS_OBJECT_TYPE_NV, 0, &hNVStoretpm);
    DBGTPM("Create NV object", resulttpm);
    resulttpm=Tspi_SetAttribUint32(hNVStoretpm,
TSS_TSPATTRIB_NV_INDEX, 0, 0x00011101);
    DBGTPM("Set index",resulttpm);
    resulttpm=Tspi_SetAttribUint32(hNVStoretpm,
TSS_TSPATTRIB_NV_PERMISSIONS, 0, 0x2000);
    DBGTPM("Set policy",resulttpm);
    // 40 bytes
    resulttpm=Tspi_SetAttribUint32(hNVStoretpm,
TSS_TSPATTRIB_NV_DATASIZE, 0, 40);
    DBGTPM("Set size",resulttpm);
    // Get policy
```

```
    resulttpm=Tspi_Context_CreateObject(hContexttpm,TSS_O
BJECT_TYPE_POLICY,TSS_POLICY_USAGE,&hNewPolicytpm);
    DBGTPM("Get TPM Policy",resulttpm);
    // Set secrt
    resulttpm=
Tspi_Policy_SetSecret(hNewPolicytpm,TSS_SECRET_MODE_PLAIN
,3,(BYTE *)"123");
    DBGTPM("Set secret 123",resulttpm);
    resulttpm                                          =
Tspi_Policy_AssignToObject(hNewPolicytpm,hNVStoretpm);
    DBGTPM("Assigned policy to NV object",resulttpm);
    // Write
    resulttpm=Tspi_NV_WriteValue(hNVStoretpm,0,18,(BYTE
*)dataToStoretpm);
    DBGTPM("Write to the NVRAM",resulttpm);
    resulttpm   =   Tspi_Context_FreeMemory(hContexttpm,
NULL);
    DBGTPM("Tspi Context Free Memory", resulttpm);
    resulttpm = Tspi_Context_Close(hContexttpm);
    DBGTPM("Tspi Context Close", resulttpm);
    return 0;
}
```

Текст программы модуля чтения из области NVRAM:

```
    int main(int argc, char **argv)
    {
        TSS_HCONTEXT hContexttpm = 0;
        TSS_HNVSTORE hNVStoretpm = 0;
        TSS_RESULT   resulttpm;
        UINT32       read_spacetpm = 18;
        char dataStoretpm[19];
```

```
        memset(dataStoretpm,0,19);
        resulttpm =Tspi_Context_Create(&hContexttpm);
        DBGTPM("Create a context", resulttpm);
        resulttpm=Tspi_Context_Connect(hContexttpm,
NULL);
        DBGTPM("Conect to TPM", resulttpm);
        resulttpm=Tspi_Context_CreateObject(hContexttpm,
TSS_OBJECT_TYPE_NV,0,&hNVStoretpm);
        DBGTPM("Creat NV object", resulttpm);
        resulttpm=Tspi_SetAttribUint32(hNVStoretpm,TSS_T
SPATTRIB_NV_INDEX,0,0x00011101);
        DBGTPM("Set index",resulttpm);
        resulttpm=Tspi_SetAttribUint32(hNVStoretpm,TSS_T
SPATTRIB_NV_PERMISSIONS,0,0x2000);
        DBGTPM("Set policy",resulttpm);
        // 40 bytes
        resulttpm=Tspi_SetAttribUint32(hNVStoretpm,TSS_T
SPATTRIB_NV_DATASIZE,0,40);
        DBGTPM("Set size",resulttpm);
        // Read NVRAM
        resulttpm=Tspi_NV_ReadValue(hNVStoretpm,0,
&read_spacetpm, (BYTE**)&dataStoretpm);
        DBGTPM("Reaad value from NV",resulttpm);
        printf("Read data: %s",&dataStoretpm[0]);
        printf(" ");
        resulttpm = Tspi_Context_FreeMemory(hContexttpm,
NULL);
        DBGTPM("Tispi Context Free Memory", resulttpm);
        resulttpm = Tspi_Context_Close(hContexttpm);
        DBGTPM("Tispi Context Close", resulttpm);
```

```
        return 0;
    }
```

Текст программы модуля удаления области NVRAM:

```
int main(int argc, char **argv)
{
    TSS_HCONTEXT hContexttpm = 0;
    TSS_HNVSTORE hNVStoretpm = 0;//Creating NVRAM space
    TSS_HPOLICY  hTPMPolicytpm = 0;
    TSS_HTPM     hTPMtpm = 0;
    TSS_RESULT   resulttpm;
    resulttpm =Tspi_Context_Create(&hContexttpm);
    DBGTPM("Create a context", resulttpm);
    resulttpm=Tspi_Context_Connect(hContexttpm, NULL);
    DBGTPM("Connect to TPM", resulttpm);
    resulttpm=Tspi_Context_CreateObject(hContexttpm,TSS_O
BJECT_TYPE_NV,0,&hNVStoretpm);
    DBGTPM("Creat an NVRAM object", resulttpm);
    resulttpm   =   Tspi_Context_GetTpmObject(hContexttpm,
&hTPMtpm);
    DBGTPM("Get TPM Object",resulttpm);
    resulttpm=Tspi_GetPolicyObject(hTPMtpm,TSS_POLICY_USA
GE,&hTPMPolicytpm);
    DBGTPM("Get Policy",resulttpm);
    resulttpm=
Tspi_Policy_SetSecret(hTPMPolicytpm,TSS_SECRET_MODE_PLAIN
,8, (BYTE *)"11111111");
    DBGTPM("Set  the  TPM  policy  secret  to  11111111",
resulttpm);
    // index = 0x00011102
```

```
    resulttpm=Tspi_SetAttribUint32(hNVStoretpm,TSS_TSPATT
RIB_NV_INDEX,0, 0x00011101);
    DBGTPM("Set NVRAM indeks", resulttpm)
    // Release NVRAM
    printf("Relese NVRAM space\r\n");
    resulttpm=Tspi_NV_ReleaseSpace(hNVStoretpm);
    DBGTPM("Release NV space", resulttpm);
    resulttpm    =    Tspi_Context_FreeMemory(hContexttpm,
NULL);
    DBGTPM("Tispi Context Free Memory", resulttpm);
    resulttpm = Tspi_Context_Close(hContexttpm);
    DBGTPM("Tspi Context Close", resulttpm);
    return 0;
}
```

Текст программы модуля чтения регистров PCR:

```
    int main(int argc, char **argv)
    {
        BYTE *rgbPcrValuetpm, *rgbNumPcrstpm;
        UINT32 ulPcrValueLengthtpm;
        UINT32 numPcrstpm, subCaptpm, i, j;
        TSS_HCONTEXT hContexttpm=0;
        TSS_HTPM hTPMtpm = 0;
        TSS_RESULT resulttpm;
        TSS_HKEY hSRKtpm = 0;
        TSS_HPOLICY hSRKPolicytpm=0;
        TSS_UUID SRK_UUIDtpm = TSS_UUID_SRK;
        // 20bytes 0
        BYTE wkstpm[20];
        memset(wkstpm,0,20);
        resulttpm =Tspi_Context_Create(&hContexttpm);
```

```
        DBGTPM("Create a context\n", resulttpm);
        resulttpm=Tspi_Context_Connect(hContexttpm,
NULL);
        DBGTPM("Connect to TPM\n", resulttpm);
        resulttpm=Tspi_Context_GetTpmObject(hContexttpm,
&hTPMtpm);
        DBGTPM("Get TPM handle\n", resulttpm);
        resulttpm=Tspi_Context_LoadKeyByUUID(hContexttpm
, TSS_PS_TYPE_SYSTEM, SRK_UUIDtpm, &hSRKtpm);
        DBGTPM("Get SRK handle\n", resulttpm);
        resulttpm=Tspi_GetPolicyObject(hSRKtpm,TSS_POLIC
Y_USAGE, &hSRKPolicytpm);
        DBGTPM("Get SRK Policy\n", resulttpm);
        resulttpm=Tspi_Policy_SetSecret(hSRKPolicytpm,
TSS_SECRET_MODE_SHA1, 20, wkstpm);
        DBGTPM("Tispi_Policy_SetSecret\n", resulttpm);
        subCaptpm = TSS_TPMCAP_PROP_PCR;
        // number of PCR
        resulttpm    =    Tspi_TPM_GetCapability(hTPMtpm,
TSS_TPMCAP_PROPERTY, sizeof(UINT32), (BYTE *)&subCaptpm,
&ulPcrValueLengthtpm, &rgbNumPcrstpm);
        if (resulttpm == TSS_SUCCESS) {
            if (ulPcrValueLengthtpm != sizeof(UINT32)) {
            printf("GetCapability(TSS_TPMCAP_PROP_PCR)
returns value != sizeof(UINT32)!");
                Tspi_Context_FreeMemory(hContexttpm,
NULL);
                Tspi_Context_Close(hContexttpm);
                exit(resulttpm);
            }
```

```
            numPcrstpm = *(UINT32 *)rgbNumPcrstpm;
            printf("\nPCR List\n");
            for (i = 0; i < numPcrstpm; i++) {
                resulttpm = Tspi_TPM_PcrRead(hTPMtpm, i,
&ulPcrValueLengthtpm, &rgbPcrValuetpm);
                printf("PCR%02u: ", i);
                for (j = 0; j < ulPcrValueLengthtpm; j++)
                    printf("%02x",  rgbPcrValuetpm[j]  &
0xff);
                printf("\n");
            }
            printf("\n");
        }
        resulttpm  =  Tspi_Context_FreeMemory(hContexttpm,
NULL);
        DBGTPM("Tispi Context Free Memory\n", resulttpm);
        resulttpm = Tspi_Context_Close(hContexttpm);
        DBGTPM("Tspi Context Close\n", resulttpm);

        return 0;
    }
```

Текст программы модуля сброса регистров PCR:

```
    int main(int argc, char **argv)
    {
        TSS_RESULT resulttpm;
        TSS_HCONTEXT hContexttpm=0;
        TSS_HTPM hTPMtpm = 0;
        TSS_HPCRS hPcrstpm;
        resulttpm =Tspi_Context_Create(&hContexttpm);
        DBGTPM("Creat a context", resulttpm);
```

```
        resulttpm=Tspi_Context_Connect(hContexttpm,
NULL);
        DBGTPM("Conect to TPM", resulttpm);
        resulttpm=Tspi_Context_GetTpmObject(hContexttpm,
&hTPMtpm);
        DBGTPM("Get TPM handle", resulttpm);
        resulttpm                                    =
Tspi_Context_CreateObject(hContexttpm,
TSS_OBJECT_TYPE_PCRS, 0, &hPcrstpm);
        DBGTPM("Creat object", resulttpm);
        resulttpm                                    =
Tspi_PcrComposite_SelectPcrIndex(hPcrstpm, 16);
        DBGTPM("Select PCR index", resulttpm);
        //Call
        resulttpm = Tspi_TPM_PcrReset(hTPMtpm, hPcrstpm);
        DBGTPM("Reset PCR", resulttpm);
        resulttpm  =  Tspi_Context_FreeMemory(hContexttpm,
NULL);
        DBGTPM("Tspi Context Free Memory", resulttpm);
        resulttpm = Tspi_Context_Close(hContexttpm);
        DBGTPM("Tspi Context Close", resulttpm);
        return 0;
    }
```

Текст программы модуля смены регистров PCR:

```
    void printMenuTPM()
    {
        printf("\nChangePCRn Help Menu:\n");
        printf("\tp PCR register to extend (0-23)\n");
        printf("\tv   value   to   be   extened   into
PCR(abcd..)\n");
```

```
        printf("\tNote: -v argument is optional and a
default value will be used if no value is provided\n");
        printf("\tExample: ChangePCRn -p 10 -v abcdef\n");
    }
    int main(int argc, char **argv)
    {
        TSS_RESULT resulttpm;
        TSS_HCONTEXT hContexttpm=0;
        TSS_HTPM hTPMtpm = 0;
        UINT32 PCR_result_lengthtpm;
        BYTE *Final_PCR_Valuetpm;
        BYTE valueToExtendtpm[250];
        int counttpm = 0;
        int pcrToExtendtpm = 0;
        memset(valueToExtendtpm,0,250);
        resulttpm =Tspi_Context_Create(&hContexttpm);
        DBGTPM("Create a context\n", resulttpm);
        resulttpm=Tspi_Context_Connect(hContexttpm,
NULL);
        DBGTPM("Conect to TPM\n", resulttpm);
        resulttpm=Tspi_Context_GetTpmObject(hContexttpm,
&hTPMtpm);
        DBGTPM("Get TPM handle\n", resulttpm);
        printf("\n Command line arguments:\n");
        for (counttpm = 0; counttpm < argc; counttpm++)
            printf("    argv[%d]    %s\n",    counttpm,
argv[counttpm]);
        if (argc >= 3)
        {
            if (strcmp(argv[1],"-p") == 0)
```

```
            {
                pcrToExtendtpm = atoi(argv[2]);
                if(pcrToExtendtpm < 0 || pcrToExtendtpm
>23)
                {
                    printMenuTPM();
                    return 0;
                }
                if (argc == 5)
                {
                    if(strcmp(argv[3],"-v") == 0)
                        memcpy(valueToExtendtpm,
argv[4], strlen(argv[4]));
                }
                else
                    memcpy(valueToExtendtpm,
"abcdefghijklmnopqrst", 20);
            }
            else
            {
                printMenuTPM();
                return 0;
            }
            //Extend value
            resulttpm                                    =
Tspi_TPM_PcrExtend(hTPMtpm,pcrToExtendtpm,20,(BYTE
*)valueToExtendtpm,

    NULL,&PCR_result_lengthtpm,&Final_PCR_Valuetpm);
            DBGTPM("Extended the PCR", resulttpm);
```

```
            Tspi_Context_FreeMemory(hContexttpm, NULL);
            Tspi_Context_Close(hContexttpm);
            return 0;
        }
        resulttpm = Tspi_Context_FreeMemory(hContexttpm,
NULL);
        DBGTPM("Tspi Context Free Memory\n", resulttpm);
        resulttpm = Tspi_Context_Close(hContexttpm);
        DBGTPM("Tspi Context Close\n", resulttpm);
        return 0;
    }
```

Текст программы модуля создания ключа подтверждения подлинности AIK:

```
    int main(int argc,char **argv)
    {
        TSS_HCONTEXT hContexttpm   = 0;
        TSS_HTPM hTPMtpm    = 0;
        TSS_RESULT   resulttpm;
        TSS_HKEY hSRKtpm      = 0;
        TSS_HPOLICY  hTPM_Policytpm = 0;
        BYTE     wkstpm[2]; //well known secret

        TSS_HKEY hMB_AIK_Keytpm=0;
        TSS_FLAG initFlagstpm;
        TSS_HPOLICY hMB_AIK_Policytpm;
        UINT32 blob_lengthtpm;
        BYTE *mb_aik_key_blobtpm;
        BYTE *pubKeytpm;
        UINT32 pubKeySizetpm;
        FILE *fouttpm;
        TSS_UUID pUuidDatatpm;
        TSS_UUID SRK_UUIDtpm   = TSS_UUID_SRK;

        memset(wkstpm,0,20);
```

```
        resulttpm = Tspi_Context_Create(&hContexttpm);
        DBGTPM("Create Context", resulttpm);
        resulttpm   =   Tspi_Context_Connect(hContexttpm,
NULL);
        DBGTPM("Context Connect", resulttpm);
        resulttpm                                        =
Tspi_Context_GetTpmObject(hContexttpm, &hTPMtpm);
        DBGTPM("Get TPM Handle", resulttpm);
        resulttpm                                        =
Tspi_Context_LoadKeyByUUID(hContexttpm,
TSS_PS_TYPE_SYSTEM, SRK_UUIDtpm, &hSRKtpm);
        DBGTPM("Get the SRK handle", resulttpm);
        resulttpm=Tspi_GetPolicyObject(hTPMtpm,
TSS_POLICY_USAGE, &hTPM_Policytpm);
        DBGTPM("GetTPM policy",resulttpm);
        resulttpm=Tspi_Policy_SetSecret(hTPM_Policytpm,
TSS_SECRET_MODE_PLAIN, 3, (BYTE *)"123");
        DBGTPM("Set TPM policy secret 123", resulttpm);
        resulttpm= Tspi_Context_CreateObject(hContexttpm,
TSS_OBJECT_TYPE_POLICY,                TSS_POLICY_USAGE,
&hMB_AIK_Policytpm);
        DBGTPM("Create   a   policy   object   for   the
key",resulttpm);

        resulttpm=Tspi_Policy_SetSecret(hMB_AIK_Policytp
m, TSS_SECRET_MODE_PLAIN, 3, (BYTE *)"123");
        DBG("Set the password to 123", resulttpm);
        // Signing key of 2048 bits
        initFlagstpm      =      TSS_KEY_SIZE_2048      |
TSS_OBJECT_TYPE_RSAKEY |
                    TSS_KEY_TYPE_SIGNING                |
TSS_KEY_AUTHORIZATION | TSS_KEY_NOT_MIGRATABLE;
        resulttpm=Tspi_Context_CreateObject( hContexttpm,
TSS_OBJECT_TYPE_RSAKEY, initFlagstpm, &hMB_AIK_Keytpm );
        DBGTPM("Ask the TPM to create the requisite key -
a non-migratable signing key, not an AIK", resulttpm);
        resulttpm=Tspi_Policy_AssignToObject(
hMB_AIK_Policytpm, hMB_AIK_Keytpm);
```

```
        DBGTPM("Assign policy to the key object",
resulttpm);
        resulttpm=Tspi_Key_CreateKey(hMB_AIK_Keytpm,
hSRKtpm, 0);
        DBGTPM("Create Attestation Identity Keys",
resulttpm);
        // Load
        resulttpm=Tspi_Key_LoadKey(hMB_AIK_Keytpm,
hSRKtpm);
        DBGTPM("Load the key into the TPM", resulttpm);
        resulttpm=Tspi_Context_FreeMemory(hContexttpm,
NULL);
        DBGTPM("Tspi Context Free Memory", resulttpm);
        resulttpm=Tspi_Context_Close(hContexttpm);
        DBGTPM("Tspi Context Close", resulttpm);
        return 0;
    }
```

Приложение Б

(обязательное)

Протокол проверки выпускной квалификационной работы в системе Антиплагиат

Протокол

проверки на оригинальность в системе «Антиплагиат ВУЗ» выпускной квалификационной работы на тему «Программное средство тестирования и управления модулем доверенной платформы, функционирующего в средствах вычислительной техники, предназначенных для обработки информации ограниченного доступа»

студента 6 курса Аксенова Алексея Александровича, направления подготовки 10.05.02 Информационная безопасность телекоммуникационных систем

программы специалитета «Разработка защищенных телекоммуникационных систем»

Руководитель ВКР инженер 2 категории АО «ПНИЭИ» Клейменова М.С.

Отчет о проверке на заимствования

Автор: Аксенов Алексей Александрович aksenoff58@yandex.ru / ID: 652
Проверяющий: Фатеев Алексей Григорьевич (ibst@pnzgu.ru / ID: 393)
Организация: Пензенский гос. Университет

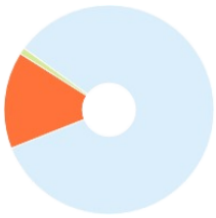
Отчет предоставлен сервисом «Антиплагиат» - http://pnzgu.antiplagiat.ru

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 39
Начало загрузки: 18.01.2018 22:50:31
Длительность загрузки: 00:00:13
Имя исходного файла: Аксенов Алексей Александрович ВКР_12ПТ1_Аксенов.docx
Размер текста: 4794 кБ
Символов в тексте: 115630
Слов в тексте: 13963
Число предложений: 449

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Последний готовый отчет (ред.)
Начало проверки: 19.01.2018 14:10:54
Длительность проверки: 00:00:19
Комментарии: не указано
Модули поиска: Модуль поиска ЭБС "БиблиоРоссика", Модуль поиска ЭБС "BOOK.ru", Коллекция РГБ, Цитирование, Модуль поиска ЭБС "Университетская библиотека онлайн", Коллекция eLIBRARY.RU, Модуль поиска ЭБС "Айбукс", Модуль поиска Интернет, Модуль поиска ЭБС "Лань", Модуль поиска "ПГУ", Кольцо вузов



ЗАИМСТВОВАНИЯ	ЦИТИРОВАНИЯ	ОРИГИНАЛЬНОСТЬ
15,27%	0,03%	84,7%

_____М.С. Клейменова