Национальный исследовательский ядерный университет
«МИФИ»

На правах рукописи

Каннер Татьяна Михайловна

Моделирование состояний аппаратной компоненты для тестирования средств защиты информации

Специальность 2.3.6 —
Методы и системы защиты информации, информационная безопасность

ДИССЕРТАЦИЯ
на соискание ученой степени
кандидата технических наук

Автор:

Научный руководитель:
кандидат технических наук, доцент
Епишкина Анна Васильевна

Москва – 2023

Оглавление

ВВЕДЕНИЕ

Актуальность темы исследования. Информационные системы (ИС) стали неотъемлемой частью жизни современного общества. При этом все больше конфиденциальной информации и персональных данных переносится в такие системы. Для обеспечения защиты данных в процессе их хранения и обработки в ИС по требованиям нормативных документов Российской Федерации необходимо использовать средства защиты информации (СЗИ): программные или программно-аппаратные. Программно-аппаратные СЗИ являются более надежными и рекомендуется применять именно их.

Достаточно часто ИС проектируются и разрабатываются без учета всех необходимых СЗИ. При этом на этапе аттестации возникает необходимость внедрения таких средств, как правило, уже разработанных, а для некоторых систем – сертифицированных. При создании же ИС с учетом средств защиты затрачиваемое время на разработку новых или адаптацию существующих СЗИ может значительно превышать время разработки самой системы. В обоих случаях для корректного использования функций безопасности и проверки отсутствия негативного влияния реализующего их СЗИ на функциональные и пользовательские характеристики системы необходимо проводить тестирование при установке таких средств в ИС. Для этого используются существующие широко известные способы и средства тестирования программного обеспечения (ПО). При этом использование существующих способов и средств тестирования ПО для функций безопасности программных СЗИ возможно без изменений, с дополнительными методическими рекомендациями по составу тестов, анализу ошибок и так далее. Это связано с тем, что среда функционирования таких средств защиты – операционная система (ОС) средства вычислительной техники (СВТ), в которой обрабатывается и хранится защищаемая информация.

Для программно-аппаратных СЗИ существуют особенности, связанные с использованием аппаратной компоненты для реализации некоторых функций безопасности. Эта компонента может, как взаимодействовать с программной компонентой СЗИ в ОС СВТ по некоторому интерфейсу и, возможно, подключаться/отключаться от СВТ в процессе работы СЗИ, так и обладать собственной средой функционирования для автономной работы относительно СВТ и его программной среды. В соответствии с этим использовать существующие способы и средства тестирования для программно-аппаратных СЗИ часто становится принципиально невозможно: реализованные в СЗИ функции безопасности нельзя проверить либо в некоторой их части, либо полностью. При этом требуется учитывать, что такие СЗИ могут иметь множество состояний как программной, так и аппартной компоненты, а также различные их сочетания. Поэтому необходимо осуществлять моделирование состояний и переходов между ними для компонент программно-аппаратного СЗИ с целью обеспечения принципиальной возможности тестирования в части всех реализованных функций безопасности.

При внедрении СЗИ в ИС, особенно сертифицированных, могут нарушаться условия функционирования этого средства вследствие влияния информационной системы на реализуемые им функции безопасности. В большей части это касается СЗИ с аппаратной компонентой, так как программные СЗИ зависят только от ОС СВТ и не зависят, например, от аппаратной платформы

и прочих подобных факторов. Из этого следует, что требуется выявлять нарушения, вносимые средой, в которой используется СЗИ, необходимых условий для выполнимости реализованных в нем функций безопасности. Такая проверка может зависеть от особенностей системы, и вследствие этого является достаточно сложной и продолжительной для каждой ИС. Кроме того, в ряде случаев время проведения всех проверок может превышать срок, по истечении которого изменятся условия тестирования функций безопасности, например, выход обновлений ОС, ИС и СЗИ. В связи с этим для своевременной фиксации нарушений функций безопасности и причин их возникновения, а также проведения тестирования в требуемые сроки необходимо автоматизировать данный процесс. Помимо этого средства автоматизации не всегда применимы к программно-аппаратным СЗИ в неизменном виде.

Таким образом, в настоящее время существует потребность в защите данных в информационных системах с применением программно-аппаратных средств защиты информации, но для проверки корректности их функций безопасности не применимы известные способы и средства тестирования программного обеспечения. В соответствии с этим требуется проведение моделирования состояний программно-аппаратных средств защиты информации и переходов между ними для разработки нового способа и средств тестирования, и тема диссертационной работы является актуальной.

Степень разработанности темы исследования. В научных трудах отечественных и зарубежных ученых и специалистов большое внимание уделяется проблемам качества и надежности ПО, а также способам и средствам его тестирования. Из таких работ необходимо прежде всего отметить труды В. В. Липаева, В. И. Грекула, И. В. Степанченко, С. В. Синицына, Н. Ю. Налютина, В. П. Котлярова, Г. Майерса (G. Myers), Р. Блэка (R. Black), Б. Бейзера (B. Beizer), Л. Тамре (L. Tamres), С. Канера (C. Kaner), Р. Калбертсона (R. Culbertson), К. Бэка (K. Beck) и Э. Дастина (E. Dusting), которые внесли наибольшее значение в развитие теории и практики тестирования.

В трудах перечисленных авторов рассматриваются способы и средства тестирования применительно к ПО. При этом вопрос применимости этих способов и средств к программно-аппаратным СЗИ в данных работах не изучается. Отличия в тестировании ПО и программно-аппаратных комплексов поверхностно рассмотрены в работах Р. Блэка, в которых обозначены сложности при проведении некоторых проверок. Однако каких-либо методических рекомендаций по тестированию программно-аппаратных средств не приводится. Таким образом, усилиями перечисленных ученых сформирована база для дальнейшего углубления и конкретизации теоретических и практических результатов для одного из актуальных на данный момент направлений исследования – тестирования функций безопасности программно-аппаратных СЗИ. Кроме того, вопросы тестирования функций безопасности программно-аппаратных СЗИ интересуют многих производителей, но описание применяемых ими на практике способов и средств тестирования обычно не публикуется в открытых источниках. Отсюда вытекает научная задача диссертации, которая состоит в формировании модели и основанного на ней способа тестирования, а также рекомендаций по практической реализации средств тестирования, устанавливаемых в информационные системы программно-аппаратных средств защиты информации. Это позволит в

условиях наличия особенностей и директивных сроков тестирования их функций безопасности обеспечить возможность проведения, полноту и оптимальность тестовых испытаний.

Целью диссертационного исследования является разработка научно-обоснованного способа тестирования функций безопасности программно-аппаратных средств защиты информации.

Задачи работы. Для достижения поставленной цели в работе решались следующие задачи:

1. Анализ известных подходов, используемых для тестирования программного обеспечения и обоснование их ограниченности в задаче полной проверки функций безопасности программно-аппаратных СЗИ.

2. Разработка научно-обоснованного способа тестирования функций безопасности программно-аппаратных СЗИ, основанного на использовании модели программно-аппаратных СЗИ и соответствующих алгоритмов тестирования и верификации их функций безопасности.

3. Разработка программно-аппаратного комплекса, реализующего предложенный способ тестирования функций безопасности программно-аппаратных СЗИ.

4. Апробация и анализ результатов применения разработанного способа тестирования и программно-аппаратного комплекса для тестирования функций безопасности программно-аппаратных СЗИ.

Положения, выносимые на защиту. В диссертационном исследовании получены и выносятся на защиту следующие положения и научные результаты:

1. Доказана ограниченность использования способов тестирования программного обеспечения в задаче полной проверки функций безопасности программно-аппаратных СЗИ, а также определены условия и критерии их применимости для тестирования программно-аппаратных средств защиты информации.

2. Разработанный способ тестирования функций безопасности программно-аппаратных средств защиты информации, основанный на использовании модели программно-аппаратных средств защиты информации, алгоритмов тестирования и верификации программно-аппаратных СЗИ, обеспечивает учет состояний аппаратной компоненты программно-аппаратных СЗИ.

3. Созданный программно-аппаратный комплекс для тестирования СЗИ, в котором реализован предложенный способ тестирования программно-аппаратных СЗИ, обеспечивает тестирование и верификацию различных видов функций безопасности.

Научная новизна работы. Новизна полученных научных результатов состоит в следующем:

1. Предложена научно-обоснованная модель программно-аппаратных СЗИ, основанная на положениях теории автоматов, в которой учитывается состояние аппаратной компоненты, что обеспечивает на основе обоснованных критериев применимости процедур тестирования проведение проверки заявленных производителем функций безопасности программно-аппаратного СЗИ, а также выявление функций безопасности, препятствующих проведению тестирования (п. 15).

2. Предложен алгоритм тестирования функций безопасности СЗИ, основанный на использовании разработанной модели программно-аппаратных СЗИ и положений теории графов, обеспечивающий решение новой задачи – тестирование программно-аппаратных СЗИ (п. 11).

3. Предложен алгоритм верификации функций безопасности программно-аппаратных СЗИ, основанный на использовании положений теории оптимизации и принятия решений, отличающийся от подобных алгоритмов использованием процедур оценки критичности выявленных в ходе тестирования ошибок и их влияния на защищенность информационной системы (п. 8, 11).

Теоретическая значимость исследования заключается в доказательстве ограниченности использования способов тестирования программного обеспечения в задаче полной проверки функций безопасности программно-аппаратных СЗИ, обосновании условий и критериев их применимости, а также разработке алгоритмов тестирования и верификации функций безопасности программно-аппаратных СЗИ.

Практическая значимость исследования заключается в разработке рекомендаций по практической реализации средств тестирования функций безопасности программно-аппаратных СЗИ, реализации на их основе программно-аппаратного комплекса для тестирования функций безопасности СЗИ, заявляемых их производителем, формулировке рекомендаций, обеспечивающих совместное использование средств тестирования функций безопасности программно-аппаратных СЗИ со сторонними вспомогательными для тестирования средствами и специализированными средствами тестирования СЗИ, что обеспечивает сокращение сроков внедрения средств защиты информации.

Методология и методы исследования. В диссертационной работе используются аппарат системного анализа и теории формальных систем, автоматов, графов, оптимизации и принятия решений.

Соответствие специальности научных работников. Содержание диссертационного исследования и полученные научные результаты соответствуют п. 8, п. 11 и п. 15 Паспорта специальности 2.3.6 Методы и системы защиты информации, информационная безопасность.

Степень достоверности научных положений и выводов обеспечена корректным использованием теорий формальных систем, автоматов, графов, оптимизации и принятия решений, строгого аппарата системного анализа, а также положительными итогами применения предложенной модели, способа, алгоритмов и разработанных средств тестирования для функций безопасности программно-аппаратных СЗИ в реализованных на практике проектах, и совпадением ожидаемых результатов от их использования с полученными при экспериментальных исследованиях.

Внедрение результатов работы. Разработанный программно-аппаратный комплекс тестирования СЗИ внедрен в ЗАО «ОКБ САПР» и применяется для тестирования функций безопасности и верификации встраиваемых в ИС программно-аппаратных СЗИ. Результаты диссертационной работы также применяются при разработке перспективных средств тестирования программно-аппаратных СЗИ в ЗАО «ОКБ САПР», при разработке программ и методик сертификационных испытаний и верификации результатов тестирования сертифицируемых средств защиты информации в ФАУ «ГНИИИ ПТЗИ ФСТЭК России», а также при создании инженерно-технических решений для высокотехнологичного производства инновационных программно-аппаратных СЗИ

на базе перспективных высокоскоростных интерфейсов информационного взаимодействия в НИЯУ МИФИ в рамках исполнения работ по НИОКТР. Аналитические результаты используются в учебном процессе кафедры «Криптология и кибербезопасность» НИЯУ МИФИ в рамках дисциплины «Программно-аппаратные средства защиты информации». Соответствующие документы, подтверждающие практическое использование и внедрение результатов исследований, приведены в приложении к тексту диссертационной работы.

Апробация работы. Основные положения и результаты работы представлены и обсуждены на следующих научных конференциях: Международная конференция «Brain-Inspired Cognitive Architectures for Artificial Intelligence», г. Москва, 2020; Международная конференция «Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology», г. Екатеринбург, 2020; Международная конференция «Intelligent Technologies in Robotics», г. Москва, 2019; Международная конференция «Безопасность инфокоммуникационных технологий», г. Воронеж, 2016; XXI Международная конференция «Комплексная защита информации», г. Смоленск, 2016; Международная конференция «Обеспечение безопасности инфокоммуникационных и цифровых технологий», г. Воронеж, 2015; XIX Международная конференция «Комплексная защита информации», г. Псков, 2014; XVIII Международная конференция «Комплексная защита информации», г. Брест (РБ), 2013; XIII Международная конференция «Информационная безопасность», г. Таганрог, 2013.

Публикации. Основные результаты по теме диссертации изложены в 21 печатной работе общим объемом 9,84 п.л., в которых автору принадлежит 7,59 п.л. Из них 16 печатных работ изданы в рецензируемых научных изданиях, определенных Высшей аттестационной комиссией при Министерстве науки и высшего образования РФ и Аттестационным советом УрФУ, и изданиях, приравненных к ним, в том числе 5 – в журналах, индексируемых международной системой научного цитирования Scopus, а также 5 – другие публикации.

Личный вклад. Содержание диссертации и основные положения, выносимые на защиту, отражают персональный вклад автора в работу. Все основные представленные в диссертации результаты получены автором самостоятельно. В работах, опубликованных в соавторстве, лично автору принадлежат: исследование применимости существующих способов и средств тестирования ПО к программно-аппаратным СЗИ; обоснование необходимости использования вспомогательных программно-технических средств для автоматизации тестирования функций безопасности программно-аппаратных СЗИ; анализ особенностей верификации программно-аппаратных СЗИ, связанных с возможностью выявления ошибок разного уровня критичности, и предложенный алгоритм верификации; исследование возможности применения различных средств автоматизации для тестирования программно-аппаратных СЗИ; классификация программно-аппаратных СЗИ и выделение особенностей тестирования различных видов функций безопасности; алгоритм тестирования функций безопасности программно-аппаратных СЗИ с использованием средств виртуализации; модель программно-аппаратного СЗИ и его представление в виде графа.

1. Анализ состояния предметной области. Постановка задач исследования

1.1. Анализ требований действующей нормативно-правовой базы в области защиты информации

Информационные системы стали неотъемлемой частью жизни современного общества. Привычные услуги, такие как запись на прием в медицинские учреждения, взаимодействие с налоговыми органами, органами внутренних дел, и так далее, в настоящее время можно получать в электронном виде через глобальную сеть Интернет. Помимо этого, большинство организаций в своей деятельности стремится использовать системы электронного документооборота и управления проектами, базы данных и тому подобное. В обоих случаях конфиденциальная информация и персональные данные пользователей хранятся и обрабатываются в ИС в цифровом виде, а для обеспечения их защиты по требованиям нормативных документов Российской Федерации необходимо использовать средства защиты информации. При этом в защите данных нуждаются не только информционные системы персональных данных (ИСПДн) и государственные информационные системы (ГИС), но и получившие в настоящее время большое распространение – значимые объекы критических информационных инфраструктур (КИИ, а также давно занявшие свою нишу в нашем мире автоматизированные системы управления производственными и технологическими процессами (АСУ ТП).

В настоящее время существует несколько нормативных документов, утвержденных приказами ФСТЭК России, регламентирующих необходимость защиты информации в таких системах. К таким документам относятся следующие:

– Требования о защите информации, не составляющей государственную тайну, содержащейся в государственных информационных системах. Утверждены приказом ФСТЭК России от 11 февраля 2013 г. №17 [91].

– Состав и содержание организационных и технических мер по обеспечению безопасности персональных данных при их обработке в информационных системах персональных данных. Утвержден приказом ФСТЭК России от 18 февраля 2013 г. №21 [85].

– Требования к обеспечению защиты информации в автоматизированных системах управления производственными и технологическими процессами на критически важных объектах, потенциально опасных объектах, а также объектах, представляющих повышенную опасность для жизни и здоровья людей и окружающей среды. Утверждены приказом ФСТЭК России от 14 марта 2014 г. №31 [90].

– Требования по обеспечению безопасности значимых объектов критической информационной инфраструктуры Российской Федерации. Утверждены приказом ФСТЭК России от 25 декабря 2017 г. № 239 [92].

Перечисленные документы содержат требования к организационным и техническим мерам защиты информации, содержащейся в ГИС, ИСПДн, АСУ ТП и значимых объектах КИИ соответственно, и согласно этим документам защита данных в таких системах является обязательной.

Для защиты данных в ИС можно применять программные или программно-аппаратные СЗИ. Программно-аппаратные средства защиты, в отличие от программных, содержат аппаратную компоненту. Аппаратная компонента полностью или частично реализует функции защиты хранящейся и обрабатываемой в ИС информации, обеспечивающие ее конфиденциальность, целостность, доступность и другие свойства ее безопасности [89] (далее – функции безопасности).

Вследствие реализации функций безопасности на аппаратном уровне, программно-аппаратные СЗИ являются более надежными. Для программных же СЗИ нельзя, например, гарантировать, что они не подверглись изменению и функционируют корректно. С помощью программных СЗИ нельзя достоверно контролировать корректность функционирования других программных средств или целостность данных, что эквивалентно неразрешимой задаче о самоприменимости [48]. Поэтому для обеспечения надежной защиты конфиденциальной информации и персональных данных при их обработке и хранении в ИС часть функций безопасности должна выполняться на аппаратном уровне, с применением программно-аппаратных СЗИ.

Информационные системы могут изначально проектироваться и разрабатываться с учетом использования СЗИ, однако часто необходимость внедрения таких средств возникает только на этапе аттестации ИС по требованиям безопасности информации РФ. В любом случае для корректного использования функций безопасности и проверки отсутствия негативного влияния реализующего их СЗИ на функциональные и пользовательские характеристики системы необходимо проводить тестирование при установке таких средств в ИС. Для этого можно использовать существующие широко известные способы и средства тестирования программного обеспечения. Однако из-за наличия аппаратной компоненты и особенностей ее функционирования эти способы и средства не всегда могут быть применимы по отношению ко всем функциям безопасности, реализованным в программно-аппаратном СЗИ.

Таким образом, в настоящий момент существует потребность в защите данных в информационных системах с применением программно-аппаратных средств защиты информации, что предполагает проведение тестирования корректности функций безопасности этих средств и отсутствия негативного влияния на ИС. Для этого необходимо вначале рассмотреть программно-аппаратные комплексы защиты информации как объекты тестирования, а также провести анализ применимости существующих способов и средств тестирования ПО по отношению к таким СЗИ.

1.2. Программно-аппаратные комплексы средств защиты информации как объекты тестирования при их установке в информационные системы

В соответствии с ГОСТ Р. 50922-2006 [17] средство защиты информации – это техническое, программное или программно-техническое средство, вещество и (или) материал, предназначенные или используемые для защиты информации.

Программные СЗИ – это средства защиты, функционирующие в некоторой среде СВТ, например, в его ОС, и реализующие всю свою функциональность на основе программного кода в этой среде без каких-либо дополнительных аппаратных средств.

Технические и программно-технические СЗИ – это средства защиты, использующие для реализации своей функциональности аппаратную компоненту. Как правило, между техническими и программно-техническими средствами защиты информации существует тонкая грань, связанная с наличием дополнительной программной компоненты, функционирующей в среде ОС СВТ. Так как в любом техническом средстве существует внутреннее ПО – прошивка или микрокод, которое управляет аппаратной компонентой, то обобщим технические и программно-технические СЗИ и будем называть их программно-аппаратными СЗИ. Также будем считать, что программно-аппаратные СЗИ состоят внутреннего и внешнего (функционирующего в ОС) ПО и реализующей необходимую аппаратную функциональность аппаратной базы.

Применяемые в настоящее время СЗИ (как программные, так и программно-аппаратные) в зависимости от реализуемых ими функций безопасности предназначены для решения совсем разных задач и делятся на соответствующие виды. Существуют средства идентификации и аутентификации, контроля целостности (КЦ), обеспечения доверенной загрузки ОС, разграничения доступа пользователей в ОС, обеспечения доверенного сеанса связи, защищенной загрузки ПО терминальных станций (ТС) по сети, криптографической защиты информации, контроля физического доступа, межсетевого экранирования, обнаружения вторжений, антивирусной защиты, а также служебные носители информации и ряд других средств защиты. Виды СЗИ по реализуемым функциям безопасности приведены на Рисунке 1.1 [12]. Средства идентификации и аутентификации осуществляют проверку субъектов доступа по некоторому идентификационному признаку и подтверждение подлинности этого субъекта на основе некой аутентифицирующей информации. Примерами таких средств являются «eToken» производства «Аладдин» и «Рутокен» – отечественной компании «Актив». Средства идентификации и аутентификации чаще всего применяются в составе или совместно со средствами доверенной загрузки и разграничения доступа пользователей в ОС.

Средства контроля целостности используются для осуществления контроля неизменности защищаемых данных и защиты от угроз целостности информации. Как правило, такие СЗИ вычисляют контрольные суммы или значения хэш-функций от защищаемых данных и реализуют проверку соответствия этих значений либо при каждом обращении к объектам, либо в заданные значения времени или при возникновении некоторых событий в соответствии с определенным регламентом. К таким СЗИ относится программа «Фикс» производства компании ЦБИ, предназначенная для вычисления контрольных сумм отдельных файлов, каталогов, контроля целостности заданного перечня файлов и так далее. Помимо этого функции контроля целостности встроены в реализацию некоторых файловых систем, например, btrfs (производства компании Oracle) и zfs (Sun Microsystems), а также используются в средствах доверенной загрузки и разграничения доступа пользователей в ОС.

СЗИ, осуществляющие функции доверенной загрузки ОС, – это, как правило, аппаратные модули доверенной загрузки (АМДЗ), которые применяются для IBM-совместимых СВТ – серверов и рабочих станций локальной сети, и обеспечивают защиту устройств и информационных ресурсов от несанкционированного доступа.

Рисунок 1.1 – Виды СЗИ по реализуемым функциям безопасности

АМДЗ перехватывает управление сразу после выполнения штатного BIOS СВТ во время процедуры POST (Power-on self-test) и позволяет выполнить дальнейшую загрузку ОС только после прохождения ряда контрольных процедур. К таким процедурам относятся [40]: идентификация и аутентификация пользователя; проверка целостности технических и программных средств СВТ с использованием пошагового контроля целостности.

АМДЗ – это СЗИ, реализующие перечисленные выше функции безопасности.

Кроме этого такие СЗИ позволяют выполнять загрузку различных ОС только с заранее определенных постоянных носителей информации, например, только с жесткого диск. Такая возможность совместно с выполнением контрольных процедур позволяет обеспечить доверенную загрузку ОС.

К средствам доверенной загрузки относятся СЗИ от несанкционированного доступа (НСД) «Аккорд-АМДЗ» производства отечественной компании «ОКБ САПР» [45] и АПМДЗ «Соболь» – компании «Код Безопасности».

Для разграничения доступа пользователей в ОС, к рабочим станциям, терминалам и терминальным серверам применяются программно-аппаратные комплексы (ПАК) средств защиты информации, среди которых можно выделить: «Аккорд-Win32», «Аккорд-Win64» [61] и «Аккорд-X» [4] производства «ОКБ САПР» (далее – ПАК «Аккорд») и «Secret Net», «Secret Net LSP» – компании «Код Безопасности».

Рассмотрим функции безопасности, выполняемые средствами разграничения доступа пользователей в ОС. Как правило, ПАК средств разграничения доступа пользователей в ОС включают в себя аппаратный модуль доверенной загрузки и функционирующее в среде ОС СПО, которые совместно реализуют функции безопасности по защите от несанкционированного доступа, идентификации и аутентификации пользователей, доверенной загрузки ОС, динамическому и статическому контролю целостности, дискреционному и мандатному механизмам разграничения достпа и т.д. [61].

Примерами СЗИ, обеспечивающими доверенный сеанс связи (ДСС), является ПАК «ViPNet Terminal» (производства компании «Инфотекс»), средство построения доверенного сеанса С-Терра «Пост» («С-Терра СиЭсПи») и средство обеспечения доверенного сеанса связи (СОДС) «МАРШ!» («ОКБ САПР») [47, 56]. Обозначим все такие средства – СОДС.

Такие комплексы предназначены для осуществления доверенного сеанса связи пользователей с доверенной информационной системой через открытый канал связи. Как правило, СОДС состоят из клиентской и серверной части, далее – Клиент ДСС и Сервер ДСС. Клиент ДСС – это, обычно, загрузочное устройство с собственным ресурсами (см. Рис. 1.2).

А Сервер ДСС – доверенный сервер, который обеспечивает создание и работу защищенного сетевого соединения с Клиентами ДСС.

Примерами СЗИ, осуществляющими защищенную загрузку ПО ТС по сети, являются решение «КАМИ-терминал» (компания «НТЦ КАМИ») и ПАК СЗИ НСД «Центр-Т» [5] («ОКБ САПР»). Защищенная загрузка ПО терминальных станций позволяет реализовать такие функции безопасности, как контроль целостности ПО ТС и обеспечение оперативного администрирования прав, назначаемых пользователям в образах ПО. Эти образы ПО ТС подписываются

Рисунок 1.2 – Клиент ДСС – ПАК СОДС «МАРШ!»

ЭП, которая проверяется перед загрузкой на терминальную станцию аппаратным клиентским устройством.

Приведенный в качестве примера программно-аппаратные комплекс «Центр-Т» реализуют следующие функции безопасности: идентификация и аутентификация администратора автоматизированного рабочего места (АРМ) «Центр» и администратора сервера хранения и сетевой загрузки (СХСЗ), генерация ключевых пар, необходимых для выработки кодов аутентификации (КА), выработка КА для образов ПО ТС, идентификация и аутентификация пользователя (перед получением ПО ТС), контроль подлинности (проверка КА) загружаемого по сети образа ПО ТС, ведение журналов загрузки ПО на ТС и журналов активности пользователей ТС.

Для криптографической защиты информации используются, например, такие средства, как: СКЗИ «КриптоПро CSP» (производства компании «КРИПТО-ПРО»), семейства комплексов «eToken» («Аладдин»), «Рутокен» («Актив») и ПСКЗИ ШИПКА [46, 64] («ОКБ САПР»).

СКЗИ могут представлять собой ПО и/или аппаратное устройство, реализующие набор криптографических функций безопасности и алгоритмов защиты информации. Для доступа к данным функциям и алгоритмам из приложений используются реализованные для них драйвера и библиотеки. Аппаратные устройства чаще всего имеют USB-интерфейс подключения к СВТ. В этом случае СКЗИ включает в себя аппаратно реализующее функции безопасности USB-устройство и специальное ПО, устанавливаемое на жесткий диск СВТ. К реализуемым такими СЗИ функциям безопасности относятся шифрование и подпись файлов или сообщений, генерация криптографических ключей и ключевых пар и т.д. [43].

В качестве служебных носителей информации применяются ПАК IronKey (производства «Kingston Technology Company») и ПАК «Секрет» («ОКБ САПР»), который реализован в 3-х вариантах исполнения [34, 38]: ПАК «Личный Секрет», ПАК «Секрет Фирмы» [55], ПАК «Секрет Особого Назначения» [44]. IronKey представляет собой защищенный шифрованный носитель информации. Комплексы «Секрет», помимо шифрования флеш-диска, реализуют функции безопасности служебного носителя и журналирования действий пользователя. Эти ПАК используются для реализации доступа к служебным носителям только на тех СВТ, на которых он разрешен для использования. Доступ к данным с других СВТ невозможен посредством архитектуры «Секрет».

Средства контроля физического доступа представляют собой программно-аппаратные комплексы безопасности, реализующие следующие функции: идентификация субъектов, ограниче-

ние и регистрация входа-выхода субъектов на заданной территории. Как правило, такие системы интегрируются с СЗИ разграничения доступа пользователей в ОС.

Межсетевой экран – это комплекс аппаратных и программных средств в компьютерной сети, осуществляющий контроль и фильтрацию проходящих через него сетевых пакетов в соответствии с заданными правилами. Основной функцией безопасности, которую реализует межсетевой экран является защита сети или отдельных ее узлов от несанкционированного доступа. Примерами таких СЗИ являются «VipNet Coordinator» и «VipNet Client» (производства компании «Инфотекс»), StoneSoft FW («McAfee») и «С-Терра Клиент» («С-Терра СиЭсПи»). Как правило, межсетевые экраны совмещены со средствами построения виртуальных частных сетей (VPN).

Система обнаружения вторжений – программное или аппаратное средство защиты, реализующее такие функции безопасности, как выявление фактов неавторизованного доступа в компьютерную систему/сеть или несанкционированного управления ими. Такие средства используются для обнаружения вредоносной активности, которая может нарушить безопасность компьютерной системы: сетевых атак против уязвимых сервисов; атак, направленных на повышение привилегий; неавторизованного доступа к файлам; действий вредоносного программного обеспечения и так далее. К средствам обнаружения вторжений относятся комплекс «Рубикон» (производства компании «НПО «Эшелон»), StoneGate IPS («McAfee»), детектор атак «Континент» («Код Безопасности»). Достаточно часто системы обнаружения вторжений и межсетевого экранирования объединяют в один продукт и выпускают совместно.

Средство антивирусной защиты представляет собой СЗИ, реализующее следующие функции безопасности: обнаружение вредоносного ПО; блокирование его работы; восстановление модифицированных вредоносным ПО файлов; предотвращение заражения файлов или операционной системы. Примером антивирусных средств защиты являются «Kaspersky Internet Security» (производства компании «Лаборатория Касперского») и «Dr.Web Антивирус» («Dr.Web»).

Систематизируя вышесказанное, для различных видов СЗИ можно составить классификационную схему, изображенную на Рисунке 1.3.

В состав ряда перечисленных выше СЗИ входит выполняющая часть функций безопасности аппаратная компонента, которая может быть реализована с использованием различных интерфейсов: для отчуждаемых устройств – USB, а для устройств, подключаемых к внутренней шине с возможным опечатыванием корпуса СВТ– PCI / PCI-express / Mini PCI-express и так далее. Также такая аппаратная компонента может представлять собой отдельный аппаратный модуль, независимый от защищаемого СВТ. Остальные рассмотренные СЗИ, например, антивирусы и другие («ФИКС», «КриптоПро CSP» и так далее) являются программными продуктами и не содержат реализующей какие-либо функции безопасности аппаратной компоненты.

Как правило, программно-аппаратные СЗИ, перехватывающие на себя управление до загрузки ОС и обеспечивающие доверенную загрузку с контролем целостности программных и аппаратных средств СВТ (например, «Аккорд-АМДЗ» и «Соболь»), представляют собой аппаратный модуль, выполненный в виде PCI-устройства. Это связано с тем, что для перехвата управления во время загрузки в современных СВТ можно использовать только расширения BIOS в виде PCI-устройств. Средства обеспечения доверенного сеанса связи (С-Терра «Пост», «ViPNet Terminal»

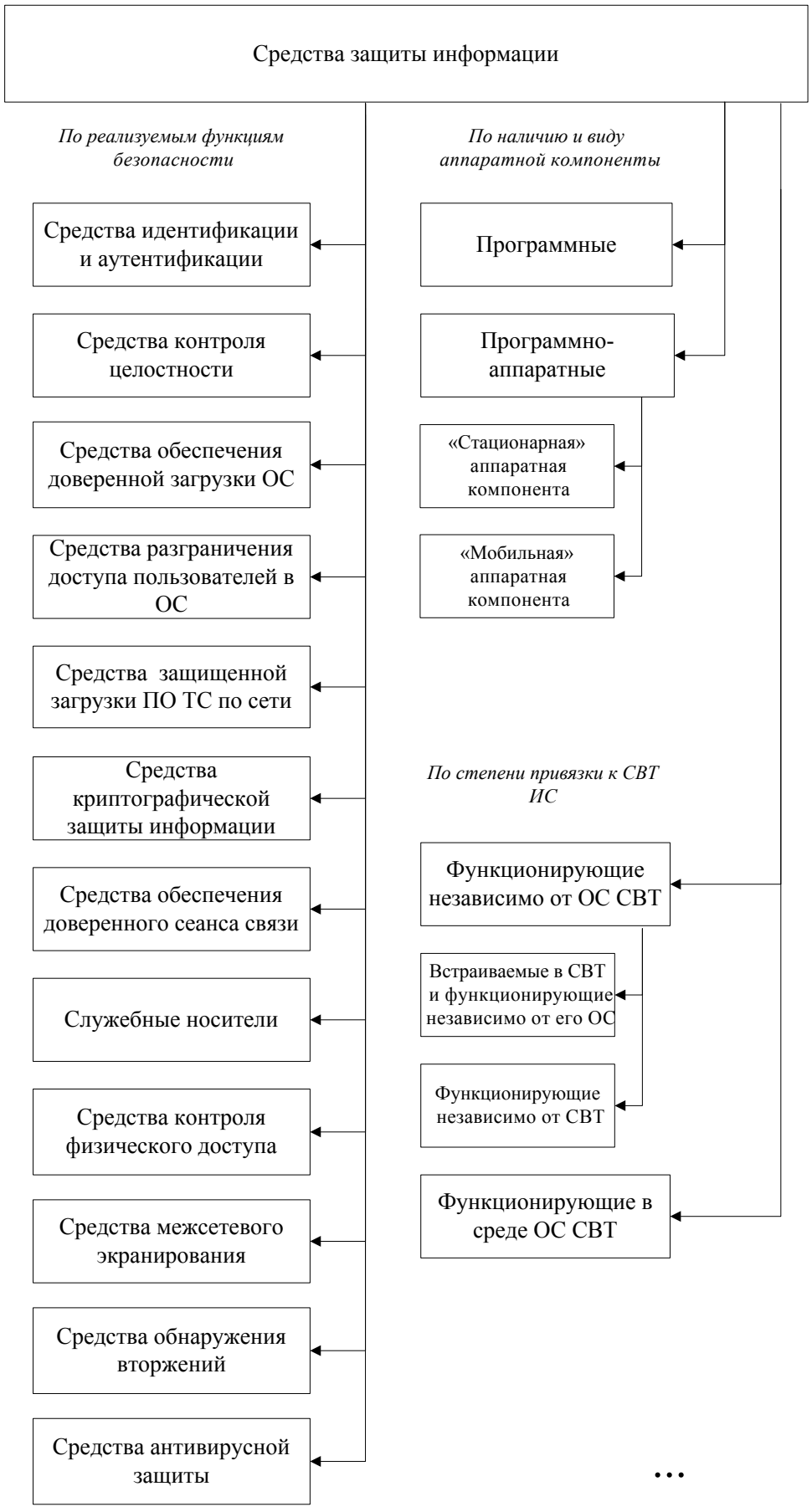


Рисунок 1.3 – Классификационная схема СЗИ

и СОДС «МАРШ!») и защищенной загрузки ПО ТС по сети («КАМИ-Терминал» и «Центр-Т») в качестве аппаратной компоненты могут иметь USB-устройства, с которых необходимо осуществлять загрузку собственной ОС СВТ средствами BIOS в соответствии с организационными мерами. К таким мерам относится установка параметров BIOS для загрузки с СЗИ в первом приоритете. Общим для этих СЗИ является независимость от ОС СВТ – вся функциональность реализуется в собственной (внутренней) ОС СЗИ совместно с его аппаратной компонентой. ОС СВТ в данном случае не используется или не загружается вообще.

Также среди рассмотренных СЗИ можно выделить средства, которые являются внешними по отношению к СВТ или «самодостаточными», то есть функционирующими независимо от СВТ. К таким средствам относятся межсетевые экраны «VipNet Coordinator» и «StoneGate» FW, детекторы атак «Рубикон» и «Континент», а также различные средства контроля физического доступа. Такие СЗИ не встраиваются в какое-либо СВТ напрямую, а работают, например, в составе локальной сети, имеют свою аппаратную базу, питание и среду функционирования программной компоненты.

У программно-аппаратных средств разграничения доступа пользователей в ОС со встроенными средствами контроля целостности аппаратной компонентой, как правило, выступает PCI-устройство (для ПАК «Аккорд» – это «Аккорд-АМДЗ», для ПАК «Secret Net» – «Соболь»). Для «eToken», «Рутокен», ПСКЗИ ШИПКА ПАК «Секрет», функционирующих в среде ОС СВТ, аппаратной компонентой является USB-устройство. Программная часть данных СЗИ функционирует в среде ОС СВТ, но основная функциональность выполняется совместно с аппаратной компонентой. Например, часть криптографических операций в «Рутокен» и «eToken», все операции в ПСКЗИ ШИПКА выполняются аппаратно, а от программной компоненты выполняется передача данных для подписи или шифрования.

Таким образом, по степени привязки к СВТ ИС программно-аппаратные СЗИ можно разделить на две группы [40, 42]:

– СЗИ, функционирующие независимо от ОС СВТ:

  – СЗИ, встраиваемые в СВТ и функционирующие независимо от ОС СВТ (либо до загрузки, либо вместо ОС СВТ);

  – СЗИ, функционирующие независимо от СВТ.

– СЗИ, функционирующие в среде ОС СВТ.

При этом аппаратная компонента программно-аппаратных СЗИ, в свою очередь, может быть одного из двух видов:

– стационарной – неизвлекаемой из СВТ в процессе функционирования СЗИ, либо представляющей собой отдельный аппаратный модуль независимый от защищаемого СВТ;

– мобильной – отчуждаемой от СВТ, то есть в процессе эксплуатации аппаратную компоненту необходимо переподключать к СВТ: физически отключать, затем подключать к соответствующему интерфейсу.

Деление по степени привязки к СВТ и по наличию аппаратной компоненты свойственны не только перечисленным выше СЗИ, но и вообще всем программно-аппаратным СЗИ. При этом все программные СЗИ могут функционировать только в среде ОС СВТ.

Для рассмотренных выше функций безопасности программно-аппаратных СЗИ можно определить классификационную схему, изображенную на Рисунке 1.4.

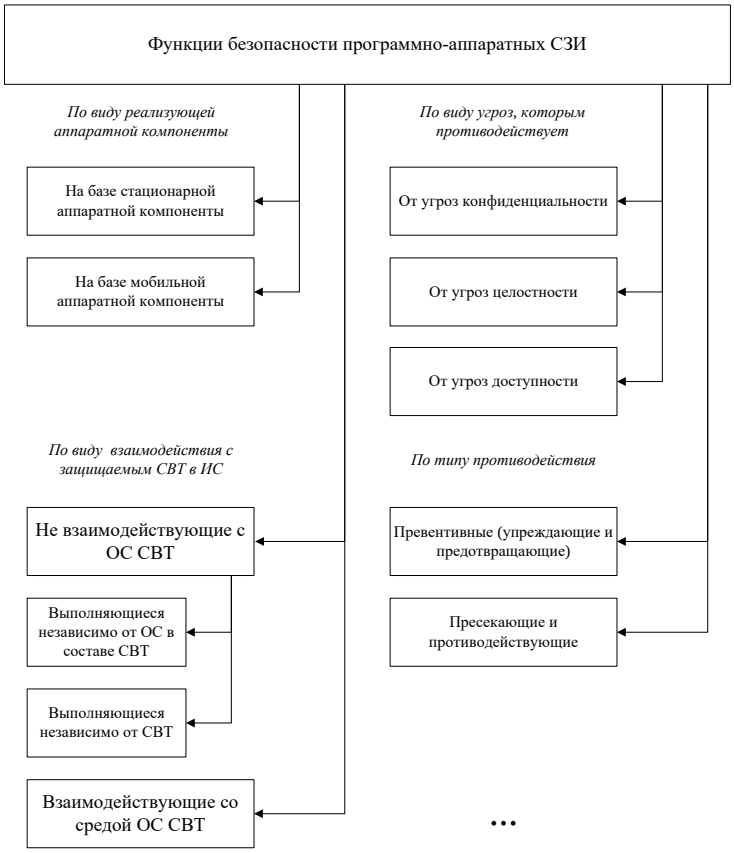


Рисунок 1.4 – Классификационная схема функций безопасности СЗИ

Функции безопасности программно-аппаратных СЗИ могут быть реализованы на базе стационарной или мобильной аппаратной компоненты, то есть иметь программно-аппаратную реализацию с частичным или полным использованием этой компоненты. Например, к функциям безопасности, реализованным на базе стационарной аппаратной компоненты относятся идентификация и аутентификация, разграничение доступа пользователей в ОС в таких СЗИ, как ПАК «Аккорд» и «Secret Net». В качестве аппаратной компоненты такие средства защиты используют PCI-устройства, неизвлекаемые в процессе работы. А к функциям безопасности, реализованным на базе мобильной аппаратной компоненты, относятся шифрование и подпись данных таких СЗИ, как «Рутокен» и ПСКЗИ ШИПКА. Перечисленные СЗИ имеют отчуждаемую аппаратную компоненту в виде USB-устройства.

По виду взаимодействия с защищаемым СВТ в ИС функции безопасности могут быть разделены на:

– Не взаимодействующие с ОС СВТ и выполняющиеся:

– независимо от ОС в составе СВТ (например, проверка целостности технических и программных средств СВТ в «Аккорд-АМДЗ» и «Соболь», а также проверка целостности ПО ТС в ПАК «Центр-Т» и «КАМИ-терминал»);

– независимо от СВТ (например, защита сети от несанкционированного доступа в «VipNet Coordinator» и StoneSoft FW, а также выявление фактов неавторизованного доступа – в системах обнаружения вторжений «Рубикон» и «Континент»).

– Взаимодействующие со средой ОС СВТ (например, хэширование и вычисление/проверка ЭП данных в «eToken», «Рутокен» и ПСКЗИ ШИПКА).

Также функции безопасности программно-аппаратных СЗИ можно разделить по виду угроз, которым они противодействуют. Бывают функции безопасности, противодействующие угрозам конфиденциальности, целостности и доступности. К первому виду таких функций относятся, например, шифрование данных в «eToken», «Рутокен» и ПСКЗИ ШИПКА, разграничение доступа пользователей в ОС в ПАК «Аккорд» и «Secret Net», а также обеспечение защищенного соединения в программно-аппаратных комплексах «Пост», «ViPNet Terminal» и СОДС «МАРШ!». Ко второму виду – контроль целостности данных в тех же ПАК и контроль целостности ПО ТС в ПАК «Центр-Т». К третьему виду относится, например, защита сети от НСД в межсетевых экранах «VipNet Coordinator» и StoneSoft FW.

Помимо этого функции безопасности рассматриваемых СЗИ бывают превентивные – упреждающие и предотвращающие, а также пресекающие и противодействующие. К превентивным функциям относятся, например, идентификация и аутентификация пользователей в «Аккорд-АМДЗ» и «Соболь», разграничение доступа пользователей ОС в ПАК «Аккорд» и «Secret Net», шифрование данных в «Рутокен» и ПСКЗИ ШИПКА, защита канала связи в комплексах «Пост», «ViPNet Terminal» и СОДС «МАРШ!», а также функция безопасности служебного носителя в ПАК «Секрет». Примерами противодействующих функций безопасности являются контроль целостности данных в ПАК «Аккорд» и «Secret Net», проверка целостности образов ПО ТС в ПАК «Центр-Т», а также проверка корректности ЭП в ПСКЗИ ШИПКА.

Вне зависимости от вида, к которому относится функция безопасности, при установке программно-аппаратного СЗИ в ИС для корректного ее использования и проверки отсутствия негативного влияния на характеристики системы необходимо проводить тестирование. При этом нужно учитывать, что функции безопасности программно-аппаратных СЗИ частично или полностью реализованы с использованием аппаратной компоненты, которая либо взаимодействует с ОС СВТ по некоторому интерфейсу (со своим прикладным ПО – программной компонентой СЗИ), либо обладает собственной средой и функционирует независимо относительно ОС СВТ и/или самого СВТ. Помимо этого в процессе функционирования, а значит, и тестирования, СЗИ может потребоваться подключение/отключение аппаратной компоненты от СВТ. Перечисленные особенности накладывают ряд ограничений на возможность применения способов и средств тестирования ПО к функциям безопасности программно-аппаратных СЗИ, а в некоторых случаях даже делают это невозможным. Также способы и средства тестирования ПО учитывают только проверку программной компоненты (ПО СЗИ) и не предусматривают необходимость проверки аппаратной компоненты и ее взаимодействия с ПО СЗИ. Указанные особенности рассматриваются производителями программно-аппаратных средств защиты, которые применяют на практике способы тестирования, как правило, реализованные для конкретного СЗИ. Однако данная информация обычно не публикуется в открытых источниках и не имеет научного обоснования.

Таким образом, реализованные в программно-аппаратном СЗИ функции безопасности невозможно проверить с использованием существующих способов тестирования ПО либо частично, либо полностью. Это первый аспект актуальности данной исследовательской работы – актуальной является задача обеспечения принципиальной возможности тестирования программно-аппаратных СЗИ в части всех реализованных функций безопасности. При этом важной является задача автоматизации процесса тестирования программно-аппаратного СЗИ. Это связано с тем, что в каждом программно-аппаратном СЗИ реализовано и используется при эксплуатации в ИС достаточно большое количество функций безопасности, каждая из которых состоит из ряда взаимодействующих между собой подфункций. Эти подфункции также требуют тестирования, как самостоятельного, так и перекрестного. Помимо этого автоматизация необходима для исключения «человеческого фактора», приводящего к ошибкам в тестировании, и для возможности организации регрессионного тестирования при дальнейшей разработке и совершенствовании СЗИ. Тестирование программно-аппаратного СЗИ в данном случае должно проводиться интегратором, владельцем ИС, совместно с разработчиком либо непосредственно при его разработке, либо в ходе проверки выполнения требуемых функций безопасности при внедрении СЗИ в ИС.

Также при внедрении любого СЗИ в ИС могут нарушаться условия функционирования этого средства. Особенно критичным этот факт является для сертифицированных средств, так как в этом случае могут быть нарушены условия действия их сертификата соответствия определенным требованиям безопасности. При этом в большей части нарушение условий функционирования касается именно программно-аппаратных СЗИ, содержащих аппаратную компоненту, так как программные СЗИ зависят только от ОС СВТ и не зависят от используемой аппаратной платформы, рабочего напряжения, типа загрузчика, версий BIOS/UEFI, прерываний и прочих подобных факторов. Все перечисленное может оказывать влияние на реализуемые программно-аппаратным СЗИ функции безопасности. Из этого следует второй аспект актуальности исследовательской работы – актуальной является также задача выявления нарушений средой (информационной системой), в которой используется СЗИ, необходимых условий для выполнимости реализованных в нем функций безопасности. Такая проверка может зависеть от особенностей системы, и за счет этого является достаточно сложной и продолжительной для каждой ИС. В связи с этим для своевременной фиксации нарушений функций безопасности и причин их возникновения необходимо максимально автоматизировать процесс тестирования в рамках тех проверок, которые в принципе могут быть автоматизированы с учетом особенностей рассматриваемой системы. Для этого необходимо использовать средства автоматизации, которые, как уже отмечалось, не всегда применимы по отношению к программно-аппаратным СЗИ в соответствии с первым аспектом.

1.3. Анализ подходов, способов и средств, используемых для тестирования программного обеспечения

В данном разделе проводится анализ существующих способов и средств тестирования программного обеспечения, к которому в том числе относятся программные средства защиты информации.

Для реализации указанной цели необходимо рассмотреть существующие понятия, подходы и способы тестирования ПО, а также – существующие средства его тестирования. Средства тестирования программного обеспечения, как правило, имеют научное обоснование и их описание можно найти во многих открытых источниках. Однако, аналогичных средств тестирования СЗИ в открытом доступе не представлено. Они, как правило, являются внутренней собственностью производителей средств защиты, и не публикуются для широкого обозрения.

Существующие подходы и способы тестирования программного обеспечения

В настоящее время при разработке любого ПО, в том числе производители все больше внимания уделяют их тестированию. Это также относится и к программным СЗИ, которые по своей сути являются ПО и отличаются от него наличием функций безопасности, нарушение которых может повлиять на безопасность защищаемой системы в целом. Тестирование нового продукта перед выпуском является одной из важных задач разработки. При этом тестирование в процессе доработки очередной версии продукта также должно являться не менее значимой задачей. В этом случае разработчики обязательно должны учитывать, что добавление новых возможностей не должно вносить новых ошибок в продукт. Также любой продукт должен проходить обязательное тестирование при его внедрении в ИС. Поэтому как и при выпуске нового и совершенствовании разработанного продукта, так и при внедрении готового продукта в ИС, тестирование должно являться неотъемлемой частью общего процесса, без выполнения которой не может быть выполнен выпуск ПО, в том числе и программного СЗИ. Таким образом, вне зависимости от того, о каком ПО идет речь: обычном прикладном ПО или о программном СЗИ, тестирование должно быть обязательной частью его жизненного цикла [21, 53, 73].

Жизненный цикл ПО — это совокупность этапов, связанных с последовательным изменением состояния программного обеспечения от принятия решения о необходимости его создания и формирования исходных требований, на всем протяжении эксплуатации конечным пользователем и до момента полного вывода ПО из эксплуатации [8, 19, 65]. В зависимости от состава этапов жизненного цикла, критериев их начала и окончания, а также их последовательности формируется модель жизненного цикла ПО. Существует несколько наиболее распространенных моделей жизненного цикла ПО: каскадная, V-образная, итерационная и спиральная [88].

В настоящее время, несмотря на большую или меньшую актуальность какой-либо модели жизненного цикла программного обеспечения, все они в той или иной мере распространены в различных компаниях. И каждая из этих моделей содержит этап (или этапы) тестирования программного обеспечения, а, следовательно, каждый производитель занимается проведением тестирования своих продуктов.

Тестирование ПО — вид деятельности в процессе разработки программного обеспечения, связанный с выполнением процедур, направленных на обнаружение и доказательство наличия в нем ошибок [57]. В процессе тестирования выполняется проверка соответствия программной реализации требованиям к ней, то есть тестирование – это управляемое выполнение программы с целью обнаружения несоответствий в ее поведении и предъявленным требованиям [27, 86].

Главная задача тестирования – определение условий, при которых появляется некорректное поведение ПО, и протоколирование этих условий. В задачи тестирования, как правило, не входит выявление конкретных неверных участков программного кода и никогда не входит исправление выявленных ошибок — это задача отладки ПО, которая выполняется по результатам тестирования [58].

Целью тестирования программного обеспечения является минимизация количества некорректных участков программного кода в конечном продукте. При этом тестирование не может гарантировать полного отсутствия таких участков в программном коде ПО. А правильно организованное тестирование дает гарантию того, что система удовлетворяет требованиям и ведет себя в соответствии с ними во всех предусмотренных ситуациях [50].

Для достижения поставленной при тестировании цели важно правильно понимать, кто должен быть задействован в этом процессе. Как правило, говоря о тестировании программного обеспечения, принято выделять две различные роли «тестировщик» и «программист», их определяют как две разные группы людей, а тестирование и программирование – как процессы, выполняемые этими двумя группами соответственно (см. независимое тестирование [54]).

Существует другой подход – программист одновременно выполняет обе роли, что в данном случае не отменяет независимого тестирования после завершения определенного этапа программирования, но учитывает то, что программист тоже должен тестировать свои программы перед передачей в другие руки [11]. Таким образом, «тестировщиком» может быть, в том числе и человек, который занимается написанием исходного кода программного обеспечения.

Однако в любом случае, даже, когда программист, по его словам протестировал выпущенную программу, следует помнить правило: каждый человек, как правило, очень бережно относится к тому, что он создавал и не хочет применять к своему «творению» какого-либо разрушительного воздействия. Так как из-за обнаружения ошибок тестирование является деструктивным процессом, то при тестировании он подсознательно будет обходить узкие места, которые могут к этому привести.

Независимо от того, кто тестирует ПО – тестировщик или программист, процесс тестирования является потенциально бесконечным, как с теоретической, так и практической стороны. Тем не менее, даже при наличии ошибок необходимо в какой-то момент времени завершить этот процесс. Решение о прекращении тестирования и выпуске продукта, как правило, принимается, исходя из соотношения «цена-качество» применительно к количеству и критичности найденных ошибок, а также срочности выпуска продукта. Чаще всего ошибкам присваивают степень (или ранг) критичности, и назначают суммарный уровень, при не превышении которого определяется готовность продукта к выпуску [39, 60, 77].

Существует несколько признаков, по которым принято проводить классификацию существующих видов тестирования ПО [7, 39, 49]: по объекту тестирования; по времени проведения; по изолированности компонентов; по позитивности сценариев; по знанию системы.

Тестирование программного обеспечения по объекту тестирования, в зависимости от преследуемых целей, можно разделить на три группы [83]: функциональное (тестирование реализуемой функциональности продукта, включая взаимодействие его компонент); нефункциональное

(нагрузочное, стрессовое тестирование, тестирование удобства использования и другие); тестирование, связанное с изменениями (регрессионное, санитарное тестирование и другие).

Существуют следующие виды функциональных тестов: непосредственно функциональное тестирование, тестирование безопасности и тестирование взаимодействия [26].

Независимо от того, какое тестирование имело место – функциональное или нефункциональное, после исправления найденных при его проведении ошибок, программное обеспечение должно быть протестировано заново для подтверждения того факта, что проблемы были действительно решены. Для этого используется еще один вид тестирования – тестирование, связанное с изменением. К такому виду тестирования относятся: регрессионное и санитарное тестирование, а также тестирование сборки [49].

Классифицируя процесс тестирования по времени его проведения можно выделить два вида: альфа- и бета-тестирование [68]. По изолированности компонентов выделяют следующие виды тестирования: модульное, интеграционное, системное, приемочное тестирование [59, 75]. Тестирование по признаку позитивности сценариев, которое предполагает проверку работоспособности ПО с использованием корректных или некорректных входных данных, разделяется на два: позитивное и негативное. Очень часто для соответствия требованиям к программному обеспечению выполняется проверка правильности обработки только корректных входных данных, и не уделяется должного внимания некорректно введенным данным. Однако нельзя забывать важность обработки программой неправильных входных данных, так как в большинстве случаев невозможно предугадать, как отреагирует на это система. Поэтому выделяют позитивное и негативное тестирование соответственно. В процессе позитивного тестирования проверяется результат работы программного обеспечения при получении им корректных входных данных. Негативное тестирование, как правило, это завершающая стадия позитивного тестирования, когда проверяется работа программного обеспечения при получении на вход некорректных данных.

Одним из вариантов негативного тестирования является способ обнаружения ошибок в ПО с использованием подачи на вход разнообразных случайных данных и наблюдение за поведением ПО (англ. fuzzing) [79].

По знанию системы тестирование делится на: белый ящик, черный ящик и серый ящик [6]. Различия в этих подходах к тестированию заключаются в тех средствах, которыми располагает тестировщик.

На основании информации из источников [7, 39, 49] можно предложить следующую классификационную схему видов тестирования, приведеную на Рисунке 1.5.

Помимо рассмотренных видов также принято выделять следующие способы тестирования: ручное тестирование; автоматизированное тестирование; автоматическое тестирование.

Каждый из этих этих способов тестирования использует ручные и/или автоматизированные тесты [9]. Вне зависимости от применяемого способа тестирования сначала необходимо выполнить разработку программы и методики испытаний (ПМИ) на основании четко определенных требований.

Рисунок 1.5 – Классификационная схема видов тестирования ПО

ПМИ – это программная документация, соответствующая ГОСТ Р. 19.301-79 [16], составляемая на основании требований, предъявляемых к программному обеспечению (или СЗИ), и содержащая следующие разделы:

– объект испытаний (наименование, область применения и обозначение);

– порядок проведения испытаний;

– требования к программе (требования, подлежащие проверке во время испытаний и заданные в техническом задании);

– требования к программной документации (состав программной документации, предъявляемой на испытания, а также специальные требования технического задания);

– состав и порядок испытаний (технические и программные средства, используемые во время испытаний, а также порядок проведения испытаний);

– способы проведения испытаний.

Ручное тестирование проводится тестировщиком вручную без использования специальных средств, в ходе тестирования выполняются следующие шаги:

– Создание ПМИ на основании технических требований (ТТ) к продукту и необходимого состава испытаний. При проведении позитивного и негативного сценария тестирования в состав и порядок испытаний ПМИ добавляются соответствующие проверки правильности обработки корректных и некорректных входных данных. В случае использования белого или серого ящика добавляются проверки исходного кода с помощью дополнительных инструментальных средств (отладчиков, дизассемблеров и статических анализаторов).

– Определение входных данных для ПМИ (например, ОС, в которых проводится тестирование; прикладное ПО, которое дополнительно установлено и так далее).

– Проведение тестирования по ПМИ, которое, в свою очередь, включает:

1. выявление некорректного поведения программного обеспечения, указывающего на наличие в нем ошибок;
2. фиксация проявления ошибок;
3. локализация зафиксированных проявлений ошибок (поиск других проявлений и взаимосвязей);
4. анализ локализованных проявлений ошибок;
5. фиксация ошибок и особенностей.

– Анализ полученных результатов тестирования.

Так как описанная последовательность действий выполняется тестировщиком вручную, то она, как правило, занимает достаточно много времени, что неприемлемо в условиях постоянного изменения требований к продукту или выхода его новых версий. Поэтому для минимизации временных затрат целесообразно внедрять и использовать автоматическое или автоматизированное тестирование.

Автоматизированное тестирование состоит из использования программ тестирования, но с частичным применением человеческого ресурса. Такой способ тестирования используется в тех случаях, когда нет возможности полностью автоматизировать все этапы тестирования и часть действий необходимо выполнять вручную. Для тестирования ПО такие ситуации достаточно редкие, и они встречаются, когда по каким-то причинам нет возможности выполнить полную автоматизацию.

Автоматическое и автоматизированное тестирование имеет ряд преимуществ по сравнению с ручным [1,22,37,42]. Вместе с тем автоматическое тестирование также имеет ряд недостатков, приведенных в [1,37,42]:

Применение автоматического или автоматизированного тестирования, как правило, наиболее целесообразно в следующих случаях [1,74]:

1. Тестирование производительности: нагрузочное, стрессоустойчивое, тестирование стабильности.
2. Регрессионное тестирование: проверка ПО на отсутствие ошибок в выпущенной версии относительно протестированной ранее предыдущей.
3. Конфигурационное тестирование: выполнение одних и тех же тестов в разных условиях. В данном случае несколько компонентов ПО требуется проверить в разном окружении, обычно заявленном в изначальных требованиях. Например, работа ПО в разных ОС.
4. Функциональное тестирование: проверка новой функциональности. Возможно использование автоматизированных тестов для последующего регрессионного тестирования.
5. Установочное тестирование: выполняется для проверки условий инсталляции и настройки продукта с учетом тех или иных требований к ПО.

При автоматизированном и автоматическом тестировании используются автоматизированные тесты – программы тестирования, разработанные при помощи специальных средств автоматизации на основании ПМИ, и повторяющие описанные в нем проверки автоматизированным способом. В этом случае последовательность шагов для способа автоматизированного тестиро-

вания соответствует последовательности шагов способа ручного тестирования, за исключением того, что тестирование проводит не человек, а запущенный им автоматизированный тест. Анализ результатов тестирования, в том числе и полученных ошибок, в любом случае проводит человек.

Автоматическое тестирование также включает использование автоматизированных тестов, но без применения человеческого ресурса. Данный способ тестирования полностью повторяет способ автоматизированного тестирования без учета действий тестировщика по запуску тестов.

При использовании ручного тестирования в ПМИ описывается перечень проверок на соответствие предъявленным требованиям, которые могут быть проведены вручную. В случае же автоматизированного или автоматического тестирования, ПМИ дополнительно включает ряд проверок, которые могут быть выполнены только с использованием автоматизированных тестов.

Функциональное, нефункциональное и связанное с изменениями тестирование, альфа и бета-тестирование, модульное, интеграционное, системное и приемочное тестирование, тестирование позитивного и негативного сценария, белого, черного и серого ящика, могут выполняться при помощи способов ручного, автоматизированного или автоматического тестирования. При этом каждый из перечисленных видов тестирования может входить в состав любого из способов тестирования, как по отдельности, так и все вместе. Вне зависимости от используемого способа тестирования необходимо изначально разработать ПМИ, содержащую требуемые состав, порядок и способы проведения испытаний. Факт использования того или иного способа тестирования отражается в ПМИ путем добавления в него соответствующих необходимых проверок. На основе ПМИ можно проводить тестирование вручную или разрабатывать автоматизированные тесты с использованием специальных средств автоматизации.

Существующие средства тестирования программного обеспечения

Как было показано в предыдущем разделе, вне зависимости от вида тестирования программного обеспечения, способ ручного тестирования требует применения программы и методики испытаний, а способ автоматизированного тестирования дополнительно к ПМИ – автоматизированных программ тестирования. При этом эти программы можно использовать как для ПО, так и для программных СЗИ, которые по своей сути являются программным обеспечением.

Для автоматизированного тестирования целесообразно использовать специальные средства, которые позволяют упростить его проведение и сократить затрачиваемое на тестирование время.

Для негативного тестирования, когда в качестве входных данных в программно обеспечение передаются заведомо некорректные данные, могут применяться специальные средства – фаззеры (англ. fuzzer), которые многократно запускают ПО (или программное СЗИ) и вводят случайные входные данные [63]. Однако разработка такого средства также требует временных затрат, так как универсального фаззера не существует. В настоящее время в ряде источников, например, в [79], представлены библиотеки и утилиты, которые могут использоваться для облегчения процесса разработки фаззеров. Все фаззеры делятся на две категории: мутационные, которые изменяют существующие образцы данных и создают условия для тестирования, и порождающие, которые каждый раз начинают тестировать с «чистого листа», моделируя необходимый протокол, формат файла или входных данных.

При тестировании с использовании подходов белого и серого ящика применяются отладчики, дизассемблеры (декомпиляторы) и, в случае необходимости, различные статические анализаторы кода [62, 69].

Тестирование белого ящика обычно предполагает использование статических анализаторов, которые позволяют проводить анализ исходного кода ПО без его реального выполнения [78]. В зависимости от используемого анализатора глубина анализа может варьироваться от определения поведения отдельных операторов до анализа, включающего весь имеющийся исходный код. С помощью статических анализаторов можно как выявлять участки программного кода, возможно, содержащие ошибки, так и математически доказать какие-либо свойства СЗИ, например, соответствие функциональности спецификации.

Дизассемблеры применяются при тестирования серого ящика для преобразования ПО в близкий к исходному код на языке программирования высокого уровня, то есть выполняется воссоздание исходного кода из объектного файла с последующим его анализом аналогично белому ящику. Отладчики используются для поиска ошибок в программном обеспечении, они позволяют выполнять трассировку, отслеживать, устанавливать или изменять значения переменных в процессе выполнения кода, устанавливать и удалять точки или условия остановки [23].

Необходимо отметить, что при использовании статических анализаторов кода, отладчиков и дизассемблеров либо тестировщик должен обладать высокой квалификацией, либо использовать указанные средства должен программист при отладке программного обеспечения перед передачей на тестирование.

Автоматизированное тестирование и автоматизированное проведение статического анализа кода предполагают использование программных тестов, которые разрабатываются при помощи специализированных средств.

В настоящее время существует достаточно большой выбор средств автоматизированного тестирования (средств автоматизации тестирования). Рассмотрим наиболее популярные из них [35]: QuickTest Professional и LoadRunner компании HP, Rational – разработки IBM, Silk – Borland, TestComplete – SmartBear Software и Sikuli – User Interface Design Group. Выбор того или иного средства автоматизированного тестирования зависит от потребностей разработчиков при создании программного обеспечения (в том числе и программного СЗИ).

Выделим основные параметры сравнения и выбора существующих на данный момент средства автоматизации [13]:

- возможность тестирования как настольных, так и web-приложений;
- возможность как функционального, так и нефункционального тестирования;
- поддержка различных версий ОС (Microsoft Windows, Linux, MacOS и других);
- работоспособность в 32-х разрядных и 64-х разрядных ОС;
- стоимость установки и эксплуатации;
- трудоемкость освоения;
- наличие технической поддержки и ее качество.

Программные СЗИ, как правило, реализованы в виде настольных или web-приложений. Средства автоматизации тестирования позволяют работать с любым классом приложений, но

при этом некоторые из них ориентированы на приложения одного класса. Поэтому при выборе средств автоматизации тестирования СЗИ необходимо учитывать специфику при работе с определенными типами приложении, исходя из конкретной задачи разработчика.

Как и во многих программных продуктах, так и в большинстве СЗИ интерфейс представлен сложными элементами (такими как иерархические списки, выпадающие меню и так далее), поэтому необходимо, чтобы средство автоматизированного тестирования имело возможность работать с такими элементами интерфейса не как с графическими образами, а как с объектами, имея доступ к их свойствам и методам. Например, такая возможность отсутствует в Sikuli.

Все приведенные выше решения, за исключением Sikuli, поддерживают как функциональное, так и нефункциональное тестирование. С использованием Sikuli можно проводить только функциональное тестирование, но при разработке СЗИ необходимо всестороннее тестирование, включая нагрузочное, стрессовое и другие виды нефункционального тестирования [109].

Большая часть существующих СЗИ функционирует в операционных системах семейства Microsoft Windows, поэтому вопрос необходимости поддержки Linux индивидуально решает разработчик в каждом конкретном случае. Из рассматриваемых средств, QuickTest Professional, LoadRunner и TestComplete поддерживают только платформу Microsoft Windows. Rational и Silk позволяют организовывать тестирование в Linux. Следовательно, если разработчику СЗИ необходима поддержка Linux, то подходят только Rational и Silk. Все средства обладают поддержкой как 32-х разрядных, так и 64-х разрядных ОС.

Рассмотрим затраты на установку и эксплуатацию. Все приведенные выше продукты, кроме QuickTest Professional и LoadRunner, принадлежат к одному ценовому диапазону. Учитывая единовременный характер трат на установку и примерно равную стоимость дальнейшей поддержки, можно прийти к выводу о равнозначности средств Rational, Silk и TestComplete. Средства QuickTest Professional и LoadRunner имеют большую стоимостью при прочих равных условиях.

При освоении конкретных продуктов важно обратить внимание на количество и удобство поддерживаемых языков для написания программ тестирования, что обеспечит снижение трудоемкости освоения конкретных средств тестирования. TestComplete предоставляет на выбор VBScript, JScript, C++Script, C#Script и DelphiScript, а в Rational доступны Java, VB.NET [100]. Из всех рассматриваемых средств автоматизированного тестирования TestComplete поддерживает наибольшее количество языков, что является его преимуществом по сравнению с остальными. В некоторых случаях при тестировании как ПО, так и СЗИ необходимо иметь возможность запуска приложений сторонних производителей и обращений к сторонним библиотекам, такие возможности предоставляют все рассматриваемые средства автоматизации.

Рассмотренные параметры сравнения выбранных средств автоматизации тестирования приведены в Таблице 1.1.

Таблица 1.1 – Сравнение средств автоматизации тестирования

Параметры сравнения	**QuickTest Professional + LoadRunner**	**Rational**	**SilkTest**	**TestComplete**	**Sikuli**
1. Возможность тестирования настольных и web-приложений	+	+	+	+	Только настольных
2. Поддерживаемые ОС	Windows	Windows, Linux	Windows, Linux	Windows, Android, iOS	Windows, MacOS X, Linux
3. Поддержка 64-х разрядных ОС	+	+	+	+	+
4. Поддержка функционального тестирования	+	+	+	+	+
5. Поддержка нефункционального тестирования	+	+	+	+	-
6. Язык программ тестирования	VBScript	Java, Visual Basic.NET	Java, JScript, VisualBasic, C#, 4Test	VBScript, JScript, DelphiScript, C++/C#Script	Jython, Python, Ruby

Во время тестирования программных СЗИ возможно возникновение критических ошибок ОС, например, «Синего экрана смерти» (BSOD). Такие ошибки приводят к прекращению работы и перезагрузке СВТ. Поэтому средство автоматизации тестирования должно уметь фиксировать моменты таких сбоев и производить возобновление выполнения тестирования после перезагрузки ОС. Все рассматриваемые средства автоматизации тестирования поддерживают возобновление работы теста после перезагрузки ОС.

При разработке СЗИ, как и другого ПО, важно поддерживать совместимость с различными аппаратными и программными платформами, поэтому при выборе средств автоматизированного тестирования необходимо учитывать возможность запуска уже разработанных тестов на различных СВТ, отличных от СВТ с установленной средой разработки тестов. Например, в средстве TestComplete предусмотрена отдельная программа для запуска тестов, не требующая установки среды разработки, для работы которой не требуется активировать лицензию.

Опыт работы со службами сопровождения и поддержки позволяет выделить производителя TestComplete [35], предоставляющего быстро реагирующую квалифицированную службу поддержки, форумы с обсуждением конкретных решений и онлайн справочную систему.

Обобщая вышеизложенное, можно прийти к выводу, что средства TestComplete, QuickTest Professional, LoadRunner, Rational и Silk применимы для автоматизации тестирования программных СЗИ. При этом такие отличия, как поддержка ОС Linux, диапазон цен и другие должны

учитываться при выборе средства автоматизации тестирования для конкретной задачи, но не являются критическими. Поэтому выбор средства автоматизированного тестирования, в большей степени, зависит от предпочтений компании-разработчика СЗИ, но существует ряд особенностей, связанных с поддержкой тех или иных технологий выбранного средства автоматизации.

При проведении тестирования ПО и программных СЗИ очень часто возникает сложность, заключающаяся в масштабируемости данного процесса. Например, требуется выполнить тестирование на большом количестве разных ОС или эмуляцию большого количества пользователей и их сеансов подключений. Для устранения этой сложности, а в некоторых случаях для удобства тестирования, могут использоваться средства виртуализации, например, производимые компаниями VMware, Oracle и Parallels, а также встроенные в ОС – Hyper-V, KVM и другие.

В результате для тестирования программных СЗИ автоматизированным способом могут применяться следующие средства тестирования:

- фаззеры (для негативного сценария тестирования);
- отладчики, дизассемблеры и статические анализаторы;
- средства автоматизированного тестирования;
- средства виртуализации.

Причем отладчики, дизассемблеры, статические анализаторы кода могут использоваться только тестировщиком, обладающим высокой квалификацией, либо программистом перед передачей программного обеспечения на тестирование.

Перечисленные средства могут использоваться для разработки и применения индивидуальных для каждого конкретного СЗИ специальных программ, необходимых для автоматизации процесса тестирования. Они не могут быть применимы непосредственно для тестирования, так как являются лишь инструментом, упрощающим его проведение.

1.4. Обоснование ограниченности известных подходов, используемых для тестирования программного обеспечения, в задаче полной проверки функций безопасности аппаратно-программных СЗИ, и необходимости модификации данных подходов, а также соответствующих способов и средств тестирования

В данном разделе обосновывается ограниченность известных подходов, используемых для тестирования программного обеспечения, в задаче полной проверки функций безопасности аппаратно-программных СЗИ, и необходимость их модификации, а также модификации соответствующих способов и средств тестирования.

Вначале исследуется возможность применения существующих способов и средств тестирования программного обеспечения к функциям безопасности программно-аппаратных средств защиты информации, функционирующих в среде ИС или независимо от нее.

Для реализации указанной цели необходимо выявить особенности программно-аппаратных СЗИ, которые могут влиять на возможность применения этих способов к таким средствам защиты. Также требуется определить необходимость, и, в случае положительного вердикта, возможность модификации существующих решений применительно к функциям безопасности программно-аппаратных СЗИ.

Помимо этого требуется определить необходимость и возможность разработки средств тестирования функций безопасности программно-аппаратных СЗИ, учитывающих выявленные особенности.

Особенности тестирования программно-аппаратных средств защиты информации

Разработка программно-аппаратных СЗИ состоит из слеудющих этапов [36]:

– проектирование (в том числе формирование требований к функциям безопасности СЗИ);

– непосредственно разработку программного обеспечения и аппаратной компоненты СЗИ (включая ее внутреннее ПО), выполняющих его функции безопасности;

– тестирование перечисленных выше компонент;

– верификацию по результатам тестирования (является частью тестирования);

– исправление найденных при тестировании ошибок;

– финализацию, то есть подготовку СЗИ к выпуску, которая состоит из фиксации версий внутреннего и внешнего ПО СЗИ, описания компенсационных мер для разрешенных по результатам верификации СЗИ ошибок, описания особенностей функционирования СЗИ, подготовки необходимой документации и тому подобного;

– выпуск СЗИ.

Этап проектирования связан с выработкой требований к функциям безопасности программно-аппаратного СЗИ. Требования к конкретному СЗИ формируются на основании задач конечного пользователя с использованием требований регуляторов в области защиты информации (ФСТЭК России и ФСБ России) к классу средства защиты, которому оно соответствует. Далее выполняется разработка программной и аппаратной компоненты СЗИ, реализующих требуемые функции безопасности. Вне зависимости от вида программно-аппаратного средства защиты, до этапа финализации и выпуска его функции безопасности должны быть проверены на соответствие предъявляемым требованиям. Соответствие требованиям проверяется не только при первом выпуске СЗИ, но и при последующих обновлениях его версий. Для этого выполняется тестирование функций безопасности программно-аппаратного СЗИ, а на его основе – верификация, включающая классификацию обнаруженных при тестировании ошибок и особенностей, а также принятие решения об успешности завершения тестирования или необходимости исправления этих ошибок [36]. Таким образом, все перечисленные этапы, кроме первого, как правило, приходится неоднократно повторять в процессе жизненного цикла программно-аппаратного СЗИ.

При проведении тестирования необходимо учитывать различные особенности, специфичные для конкретного вида СЗИ и функций безопасности (см. раздел 1.2). Программные СЗИ представляют собой ПО, функционирующее в среде ОС, поэтому для них можно использовать все описанные ранее правила, виды и способы тестирования. Единственной особенностью в этом случае является необходимость проведения проверок не только штатного функционирования СЗИ, как в случае с ПО, но и моделирование «атак» на систему, защиту которой выполняет данное СЗИ. Перечисленные проверки требуется включать в ПМИ. В отличие от ПО у программных и программно-аппаратных СЗИ при тестировании могут быть обнаружены как ошибки ин-

терфейса и ошибки, ограничивающие функциональность, так и ошибки, способные повлиять на безопасность информации в защищаемой системе из-за нарушения функций безопасности СЗИ. Последние являются наиболее критичными ошибками, наличие даже одной ошибки такого типа обычно требует отправки СЗИ на доработку.

Программно-аппаратные СЗИ, в отличие от программных, имеют больше особенностей, возникающих при их тестировании. Это связано с тем, что частично или полностью функции безопасности таких СЗИ реализованы с использованием аппаратной компоненты – нужно тестировать не только программную компоненту (ПО СЗИ), но и функциональность аппаратной компоненты и ее взаимодействие с ПО СЗИ.

При тестировании аппаратной компоненты программно-аппаратного СЗИ необходимо различать специфичные и неспецифичные части этой компоненты, а именно: компоненты, непосредственно реализующие функции безопасности (микропроцессор, внутреннее ПО, ПО в ОС); прочие компоненты, не реализующие функции безопасности напрямую (флеш-память и другие).

Тестирование программно-аппаратного СЗИ в части функций безопасности носит специализированный, а тестирование прочих компонент – универсальный характер, то есть для них существует возможность использования стандартных средств тестирования [38, 41]. При этом, кроме тестирования таких компонент по отдельности, необходимо проводить тестирование их совместимости и взаимодействия, то есть перекрестное тестирование. Поэтому в программу и методику испытаний должны включаться соответствующие проверки как для компонент, реализующих функции безопасности, так и для прочих компонент.

Функции безопасности программно-аппаратных СЗИ, реализуемые в аппаратной компоненте хранятся в виде исполняемого кода во внутренней памяти СЗИ, чаще всего во флеш-памяти NAND или EEPROM. Поэтому от работоспособности памяти СЗИ зависит корректность выполнения функций безопасности. В связи с этим необходимо обязательно проводить тестирование по определению и подтверждению скоростных характеристик данной памяти, а также ее нагрузочные и другие тесты. Такие проверки должны быть обязательно добавлены в программу и методику испытаний. При этом следует отметить, что тестирование памяти в аспекте определения ее надежности провести вручную можно, но вследствие необходимости выполнения большого количества циклов атомарных операций чтения/записи различных по объему блоков данных, повлечет за собой достаточно большие временные затраты. Для тестирования памяти целесообразно применять существующие и широкодоступные специализированные средства, вместо разработки собственных программ тестирования [38, 41].

Наличие аппаратной компоненты также требует проведения тестирования взаимодействия функций безопасности СЗИ с различными аппаратными и программными платформами. Это связано с тем, что функционирование аппаратной компоненты может зависеть от множества факторов – совместимости по рабочему напряжению, с различными ОС, загрузчиками, BIOS/UEFI, прерываниями и так далее. Для каждого вида аппаратной компоненты и СЗИ характерна часть или все из этих факторов [39, 84].

При проведении нефункционального тестирования имеет место проблема масштабируемости, основанная на том, что достаточно сложно смоделировать реальную систему, например,

при нагрузочном или стрессовом тестировании. В случае с программными СЗИ, могут использоваться средства виртуализации, с помощью которых можно сэмулировать большое количество пользователей и их сеансов подключений. Для программно-аппаратных СЗИ изначально необходимо использовать реальные СВТ, что усложняет организацию процесса тестирования. Также не все программно-аппаратные СЗИ можно тестировать с использованием средств виртуализации из-за невозможности перенаправления аппаратной компоненты в виртуальную среду. При нагрузочном тестировании программно-аппаратных СЗИ необходимо уделять внимание характеристикам аппаратной компоненты – разводка платы, рабочее напряжение, а также следить за нагревом устройств и так далее [39].

При конфигурационном тестировании программно-аппаратных СЗИ необходимо тестировать все возможные варианты аппаратной компоненты и все сочетания с программной компонентой, так как аппаратная компонента может быть реализована на базе разных вариантов исполнений, функционирование одного из них может отличаться от работы других.

Модульное тестирование на первом этапе должно проводиться для программной и аппаратной компонент по отдельности, и только затем можно переходить к тестированию интеграции этих модулей между собой. Системное тестирование, в свою очередь, должно выполняться только после интеграционного.

Аппаратная компонента часто состоит из двух «слоев» – аппаратного устройства с внутренним программным обеспечением и драйвера его взаимодействия с ОС. Поэтому модульное тестирование распадается на два подуровня, на одном из которых тестируется драйвер, а на другом – само аппаратное устройство. Поэтому тестирование отдельных функций программно-аппаратного СЗИ может быть выполнено только на интеграционном уровне, а не на модульном, как у программных СЗИ [39].

Применение подходов белого и серого ящика к тестированию программно-аппаратных СЗИ, функционирующих независимо от ОС СВТ, затруднено из-за особенностей анализа кода и необходимости встраивания средств тестирования в ОС СЗИ.

Как было показано в разделе 1.2, функции безопасности программно-аппаратных СЗИ классифицируются по следующим признакам: по виду реализующей аппаратной компоненты; по виду взаимодействия с защищаемым СВТ в ИС; по виду угроз, которым противодействует; по типу противодействия. Рассмотрим особенности тестирования функций безопасности программно-аппаратных СЗИ с точки зрения таких признаков.

Для классификации функций безопасности программно-аппаратных СЗИ по виду взаимодействия с защищаемым СВТ в ИС (не взаимодействующие с ОС СВТ и взаимодействующие со средой ОС СВТ) существует наибольшее количество таких особенностей.

Функции безопасности, не взаимодействующие с ОС СВТ, реализуются в следующих программно-аппаратных СЗИ:

– встраиваемых в СВТ и функционирующих независимо от ОС СВТ (либо до загрузки, либо вместо ОС СВТ);

– функционирующих независимо от СВТ.

Функции безопасности, не взаимодействующие с ОС СВТ и выполняющиеся независимо от него, реализуются в программно-аппаратных СЗИ, представляющих собой самодостаточные устройства с собственным питанием, процессором и памятью. Функции безопасности, не взаимодействующие с ОС СВТ и выполняющиеся независимо от нее в составе СВТ, реализуются в программно-аппаратных СЗИ, либо перехватывающих на себя управление до загрузки ОС, либо с которых необходимо инициировать загрузку до (вместо) ОС СВТ. Аппаратная компонента таких СЗИ может представлять собой аппаратный модуль, выполненный чаще всего в виде PCI- или USB-устройства. Общим для всех этих СЗИ является независимость от ОС СВТ – вся функциональность, в том числе и функции безопасности, реализуется в собственной внутренней ОС СЗИ совместно с его аппаратной компонентой. ОС СВТ при работе таких СЗИ не используется или не загружается вообще.

При применении любого способа тестирования к функциям безопасности программно-аппаратных СЗИ, не взаимодействующим с ОС СВТ и выполняющимся независимо от нее в составе СВТ, необходимо обязательно учитывать следующие особенности [40, 41]:

– Функции безопасности выполняются в среде ОС СЗИ. Поэтому при проведении тестирования возникает сложность фиксации ошибок функционирования средства защиты. Это связанно либо с невозможностью записи большого объема информации в память СЗИ, либо с тем, что любое возникновение серьезных ошибок, как правило, приводит к перезагрузке СВТ без сохранения каких-либо результатов. При этом возможны оба случая одновременно. Из-за этого возникает сложность в интерпретации результатов тестирования – остаются неизвестными момент времени и место появления ошибки.

– СЗИ может быть реализовано на базе аппаратных компонент с различным исполнением (например, для СЗИ НСД «Аккорд-АМДЗ» – Аккорд-5МХ, 5.5, 5.5e, GX, GXM, GXMH). В этом случае входные данные ПМИ необходимо дополнять различными вариантами аппаратной компоненты. Это приводит к тому, что объемы проверок функций безопасности увеличиваются пропорционально количеству исполнений аппаратной компоненты, и такие проверки становится сложно провести вручную. Также каждый из вариантов аппаратной компоненты может иметь отличающиеся входные условия функционирования и тестирования, например, различный порядок действий пользователя при идентификации и аутентификации. Поэтому при тестировании одной и той же функции безопасности, реализованной с помощью различных аппаратных компонент, могут понадобиться некоторым образом измененные в зависимости от исполнения аппаратной компоненты тесты. Это также должно быть обязательно отражено в ПМИ.

– Существует необходимость проведения перекрестного функционального тестирования составляющих программно-аппаратного СЗИ, реализующих функции безопасности, то есть тестирования взаимодействия. Например, надежности интерфейса управления, надежности каких-либо аппаратных компонент – внутренней памяти и так далее, что должно быть обязательно учтено в ПМИ. Необходимость перекрестного тестирования приводит к увеличению количества проверок и, из-за большого их объема, тестирование вручную становится затруднительным.

– Существует зависимость возможности выполнения функций безопасности аппаратной компоненты СЗИ от аппаратной платформы, то есть необходимо проводить тестирование на раз-

нообразных аппаратных платформах с различными версиями BIOS/UEFI, различными прерываниями и питанием [84]. В этом случае входные данные ПМИ необходимо дополнить проверками на различном оборудовании. Однако автоматизировать тестирование функций безопасности, реализованных на базе СЗИ, функционирующих независимо от ОС СВТ на всем разнообразии оборудования затруднительно, так как необходимо вручную устанавливать аппаратную компоненту в СВТ и/или настраивать параметры BIOS и тому подобное. Поэтому автоматизированные тесты для функций безопасности таких программно-аппаратных СЗИ приходится дополнять ручной проверкой работоспособности на различных аппаратных платформах. Также нужно учитывать, что тестирование на одной или небольшом количестве аппаратных платформ не гарантирует работоспособности функций безопасности СЗИ на всех возможных платформах.

– Для тестирования функций безопасности аппаратной компоненты СЗИ можно использовать программный интерфейс, реализующий все функции взаимодействия программной компоненты с аппаратной. Данный интерфейс целесообразно применять именно в среде ОС СЗИ, так как в среде ОС СВТ в общем случае нельзя проверить все функции безопасности СЗИ, реализованные в аппаратной компоненте, например, как АПМДЗ «Соболь» перехватывает загрузку ОС СВТ. Это также приводит к затруднению интерпретации результатов тестирования. При этом с помощью данного интерфейса можно автоматизированно эмулировать действия пользователя, проверять предельные/граничные значения, однако, дополнительно необходимо проводить тестирование программной компоненты, чтобы убедиться что она также функционирует корректно.

– Существует возможность тестирования некоторых функций безопасности программно-аппаратных СЗИ, реализованных в программной компоненте, которая может быть запущена на выполнение не в среде ОС аппаратного устройства, например, для «Аккорд-АМДЗ» – контроллера Аккорд, а в среде аналогичной ОС, установленной на СВТ. В рамках ОС СВТ можно автоматизировать процесс проверки корректности исполнения различных модулей, провести нагрузочное и стрессовое тестирование, то есть интерфейса взаимодействия с аппаратной компонентой и тому подобное, проверить корректную работу программной компоненты. Но в данном случае не будет гарантирована корректность работы функции безопасности у конечного пользователя в финальном продукте, так как программная компонента должна так же корректно выполняться и в среде ОС СЗИ.

При тестировании функций безопасности программно-аппаратных СЗИ, не взаимодействующих с ОС СВТ и выполняющихся независимо от СВТ, также необходимо учитывать рассмотренные выше особенности, за исключением одной – в данном случае нет зависимости от аппаратной платформы СВТ, в качестве нее выступает аппаратная компонента самого СЗИ.

Рассмотренные особенности приводят к тому, что для тестирования функций безопасности программно-аппаратных СЗИ, не взаимодействующих с ОС СВТ, нельзя использовать существующие способы тестирования ПО без внесения в них изменений – требуется их дополнение, а, следовательно, формирование нового способа тестирования.

Функции безопасности программно-аппаратных СЗИ, взаимодействующие со средой ОС СВТ, могут быть реализованы как в ПО, устанавливаемом в этой среде, так и в аппаратной

компоненте. Необходимо отметить, что ключевые функции безопасности при этом выполняются в аппаратной компоненте. Например, все криптографические операции в ПСКЗИ ШИПКА выполняются аппаратно, а от программной компоненты только передаются данные, которые необходимо подписать или зашифровать. Как правило, аппаратная компонента в этом случае представляет собой PCI- или USB-устройство.

При применении любого способа тестирования к функциям безопасности программно-аппаратных СЗИ, взаимодействующим со средой ОС СВТ, необходимо обязательно учитывать следующие особенности [41, 42], которые частично повторяют перечисленные выше, но имеют свои нюансы:

– Функции безопасности СЗИ могут быть реализованы в аппаратной компоненте, имеющей различные варианты исполнения. Например, ПСКЗИ ШИПКА может быть реализовано на базе ШИПКА-лайт, ШИПКА-лайт Slim, ШИПКА-1.6 и ШИПКА-2.0. При этом, в отличие от СЗИ, функционирующих независимо от ОС СВТ, каждая из таких аппаратных компонент поразному взаимодействует с программной компонентой – ПО, устанавливаемым в ОС. Для таких СЗИ существует зависимость взаимодействия функций безопасности, реализованных в различных исполнениях аппаратной компоненты с операционной системой, в которой функционирует это средство защиты. Поэтому при проведении тестирования ПМИ необходимо дополнять различными вариантами исполнения аппаратной компоненты, входными данными и проверками, специфичными именно для нее, а также проверками взаимодействия с различными ОС. Увеличение количества проверок приводит к целесообразности автоматизации процесса тестирования.

– Как и для функций безопасности, не взаимодействующих с ОС СВТ, существует необходимость проведения перекрестного функционального тестирования аппаратной компоненты и ПО (установленного в ОС), которые совместно реализуют функции безопасности программно-аппаратного СЗИ. Однако в данном случае перечень этих проверок так же зависит от ОС, установленной в СВТ. Это должно быть добавлено в ПМИ, количество проверок в данном случае становится пропорционально количеству ОС, в которых может функционировать СЗИ.

– При нагрузочном и стрессовом тестировании функций безопасности СЗИ, необходимо проверять, в том числе предельные значения его характеристик [42]. В отличие от функций безопасности, не взаимодействующих с ОС СВТ, данные проверки для программной компоненты проводятся с меньшими ограничениями (вследствие функционирования в ОС СВТ, а не СЗИ), а для аппаратной – при тех же ограничениях. Например, для проверки корректности функционирования ПСКЗИ ШИПКА с максимально возможным количеством криптографических ключей (которое может быть более 200) нужно сгенерировать их необходимое количество, проверить корректность функционирования СЗИ, в случае возникновения ошибок – поэтапно уменьшать количество ключей и повторять тестирование до определения критического значения. Эти проверки также должны быть добавлены в ПМИ. Проводить такое тестирование также целесообразно автоматическим способом.

– Для функций безопасности программно-аппаратных СЗИ, взаимодействующих со средой ОС СВТ, как правило, реализованы комплекты разработчика (англ. Software Development Kit, SDK). SDK реализуется для стандартных интерфейсов, предоставляемых СЗИ для вызова

его функций безопасности из ОС. Например, для «eToken», «Рутокен» и ПСКЗИ ШИПКА в целях возможности использования функций безопасности сторонними разработчиками в настоящее время реализован ряд стандартных интерфейсов. При этом SDK включает набор программ тестирования по использованию перечисленных интерфейсов, которые можно применять для автоматизации процесса тестирования функций безопасности СЗИ. Успешное выполнение этих примеров гарантирует, что функции безопасности аппаратной компоненты, используемые в утилитах, работающих на базе перечисленных выше интерфейсов, будут также работать корректно. Однако дополнительно необходимо проводить тестирование программной компоненты СЗИ, работа которой не проверяется при использовании SDK [37].

В отличие от функций безопасности программно-аппаратных СЗИ, не взаимодействующих с ОС СВТ, для рассматриваемых функций безопасности существует возможность фиксации ошибок как программной, так и аппаратной компоненты в операционной системе СВТ.

Помимо этого для функций безопасности аппаратных компонент, реализованных на базе USB-устройств, функционирующих в ОС СВТ, нет необходимости проведения тестирования на разнообразных аппаратных платформах. Это связано с отсутствием ограничений на аппаратные характеристики СВТ, достаточно соответствия спецификации определенной версии интерфейса USB.

Также вследствие отсутствия ОС СЗИ, функции безопасности программно-аппаратных СЗИ, взаимодействующие с ОС СВТ, изначально тестируются в среде операционной системы (а не в среде СЗИ).

Для для функций безопасности программно-аппаратных СЗИ, взаимодействующих со средой ОС СВТ, существует меньше особенностей (чем для не взаимодействующих с ОС СВТ), однако, для них также требуется сформировать соответствующий новый способ тестирования.

Рассмотрим особенности тестирования функций безопасности программно-аппаратных СЗИ с точки зрения классификации по виду реализующей аппаратной компоненты: на базе стационарной или мобильной аппаратной компоненты.

При автоматизации тестирования функций безопасности программно-аппаратных СЗИ, реализованных при помощи мобильной (отчуждаемой) аппаратной компоненты, возникает особенность, связанная с необходимостью переподключения аппаратной составляющей в процессе тестирования используемой конечными пользователями функциональности. Такие проверки необходимо учитывать в ПМИ и автоматизированных тестах. Для функций безопасности, реализованных в стационарной аппаратной компоненте (неизвлекаемой из СВТ в процессе функционирования СЗИ), нет необходимости проводить проверки, присутствующие при возможности отчуждения аппаратной компоненты пользователем. Однако, в данном случае, если возникает необходимость переноса стационарной аппаратной компоненты, например, для установки в другое СВТ и проверки взаимодействия с различными аппаратными платформами. Процесс тестирования функций безопасности в данном случае требует дополнительных действий, например, может потребоваться распломбировать корпус СВТ, открутить крепежные болты, снять крепежи, извлечь контроллер, перенести его в другое СВТ, а затем выполнить те же действия в этом СВТ в обратном порядке. Поэтому использовать существующие способы тестирования программного

обеспечения для функций безопасности, реализованных в стационарной и в мобильной аппаратных компонентах, не представляется возможным, так как стационарную компоненту необходимо устанавливать единожды (а ее перенос в другое СВТ рассматривать отдельно), а мобильную – подключать/переподключать к СВТ во время функционирования СЗИ.

Помимо рассмотренных выше для всех видов программно-аппаратных СЗИ существует общая особенность: необходимо проводить анализ влияния обнаруженных при выполнении реализованных функций ошибок на безопасность защищаемой ИС. Такие проверки должны быть добавлены в программу и методику испытаний. Эта особенность относится ко всем СЗИ (не только программно-аппаратным), но не является характерной для ПО или любых других программно-аппаратных средств. С учетом этого для средств защиты, в том числе и программно-аппаратных, имеют место особенности процесса верификации [36].

В общепринятом понимании верификация – это подтверждение соответствия выпускаемого продукта предъявляемым к нему требованиям [52]. Принятие решения об итогах верификации выполняется на основании проведения комплексного тестирования программно-аппаратного СЗИ, потому что именно тестирование приводит к выявлению всех ошибок и недочетов его работы, касающихся как удобства использования, так и нарушения работоспособности. При этом вывод о соответствии функций безопасности тестируемого СЗИ заявленным требованиямна делается на основе анализа результатов выполненного тестирования. То есть тестирование и верификация – это два взаимосязанных процесса: результаты верификации могут привести к началу очередного тестирования, а полученные результаты тестирования являются основой верификации.

Тестирование и верификация СЗИ состоят из следующих шагов [36]:

1. Тестирование:
   – выявление некорректного поведения СЗИ, указывающего на наличие в нем ошибок (в том числе и в функциях безопасности);
   – фиксация проявления ошибок;
   – локализация проявлений ошибок – поиск других проявлений и взаимосвязей;
   – анализ локализованных проявлений ошибок;
   – фиксация ошибок и особенностей.
2. Верификация:
   – классификация ошибок и особенностей: определение типа, возможности компенсации, степени критичности;
   – принятие решения об итогах верификации и о возможности финализации СЗИ или возвращении на доработку с последующим повторным тестированием.

Как было показано ранее, в процессе тестирования программно-аппаратных СЗИ используются программы и методики испытаний, содержащие в себе определенные последовательности действий и описания ожидаемых результатов, учитывающие особенности функционирования конкретного СЗИ. ПМИ составляются таким образом, чтобы охватить всю функциональность продукта и получить наиболее полное представление о его работоспособности. При этом тестируемый объект может находиться в различных начальных условиях (разные ОС, разные ва-

рианты аппаратной компоненты и тому подобное), которые являются входными данными для проведения тестирования. Для каждого такого набора условий тестирование по ПМИ выполняется отдельно. На основании каждого проведенного цикла тестирования составляется таблица результатов, содержащая перечень всех выявленных проявлений ошибок. После этого тестировщик проводит локализацию ошибок путем исследования их выявленных проявлений и анализа возможных взаимосвязей с ранее встречавшимся неверным поведением продукта, если таковое было. На основании полученных от тестировщика данных программист устанавливает ошибку и сообщает тестировщику, какие функции может затрагивать обнаруженная ошибка. Тестировщик, в свою очередь, проверяет, оказывает ли данная ошибка влияние на указанные функции продукта, а затем формирует окончательный список найденных ошибок и особенностей [36]. Однако, для программно-аппаратных СЗИ необходимо понимать оказывает ли обнаруженная ошибка влияние на безопасность защищаемой ИС в целом. Существеющие подходы верификации, например, вероятностный подход [24, 95], не дают ответа на данный вопрос, а лишь проводит анализ вероятности отказа ИС.

Таким образом, особенности тестирования функций безопасности программно-аппаратных СЗИ возникают при их рассмотрении с точки зрения двух первых признаков классификаций (по виду реализующей аппаратной компоненты и виду взаимодействия с защищаемым СВТ в ИС), деление же по двум последним признакам (по виду угроз, которым противодействуют и по типу этого противодействия) не коррелирует с особенностями программно-аппаратных СЗИ, отличающими их от программного обеспечения. В отличие же от программно-аппаратных СЗИ, функции безопасности программных средств защиты нельзя классифицировать по виду реализующей аппаратной компоненты и виду взаимодействия с защищаемым СВТ в ИС, так как программные СЗИ не имеют такой компоненты и всегда функционируют с среде ОС СВТ. Помимо этого существуют особенности программно-аппаратных СЗИ, которые не учитываются при верификации ПО. Наличие рассмотренных особенностей может повлиять на принципиальную возможность проведения тестирования и на его результаты, а это значит, что существующие способы тестирования применительно к функциям безопасности программно-аппаратных СЗИ требуют совершенствования. А значит, требуется разработка научно-обоснованного способа тестирования функций безопасности программно-аппаратных СЗИ.

Особенности применения средств тестирования программного обеспечения для программно-аппаратных средств защиты информации

Так как программные СЗИ представляют собой ПО, функционирующее в ОС, то для них можно использовать все описанные выше средства тестирования. Для программно-аппаратных СЗИ различных видов применение рассмотренных средств тестирования может быть затруднено из-за наличия аппаратной компоненты. Это связано с тем, что функции безопасности программно-аппаратных СЗИ частично или полностью реализованы с использованием аппаратной компоненты, поэтому применять средства тестирования необходимо не только для программной компоненты (ПО СЗИ), но и для аппаратной.

Для тестирования программно-аппаратных СЗИ, функционирующих в среде СВТ ИС, как правило, с некоторыми ограничениями можно использовать большинство средств тестирования программных СЗИ. Однако для СЗИ, функционирующих независимо от ОС, использование таких средств может быть затруднено. Это связано с тем, что некоторые программно-аппаратные СЗИ могут функционировать как встраиваемые устройства (англ. embedded device), то есть могут иметь свой микропроцессор, память и ОС. Поэтому необходимо проводить встраивание средств тестирования в ОС СЗИ, что по ряду причин не всегда возможно.

Рассмотрим особенности применения существующих средств тестирования для функций безопасности программно-аппаратных СЗИ с точки зрения их классификации, выполненной в разделе 1.2.

При применении средств тестирования к функциям безопасности программно-аппаратных СЗИ, не взаимодействующим с ОС СВТ и выполняющимся независимо от нее в составе СВТ, необходимо обязательно учитывать следующие особенности [40, 41]:

– Использование статических анализаторов и дизассемблеров для рассматриваемых функций безопасности возможно путем переноса анализируемого исходного кода из ОС СЗИ в ОС СВТ. В этом случае необходимо учитывать, что анализируемый код должен исполняться в условиях ограниченных ресурсов в среде ОС СЗИ, а не в ОС СВТ, где таких ограничений нет.

– Функции безопасности реализованы и выполняются в среде ОС СЗИ, поэтому, в общем случае, для них невозможно использовать фаззеры, отладчики и средства автоматизированного тестирования, а также разработанные с их использованием программы тестирования. Это связано с тем, что эти средства тестирования также необходимо встраивать в ОС СЗИ, при этом может вообще отсутствовать возможность применения таких средств в ОС СЗИ. При невозможности использования средств тестирования, нельзя использовать и программы тестирования. Если же такая возможность существует, то возникает ряд сложностей, связанных с фиксацией результатов работы этих средств, так как программно-аппаратные СЗИ могут запрещать доступ к носителям информации СВТ и иметь недостаточный объем собственной памяти. При этом возникновение серьезных ошибок в процессе функционирования средств тестирования будет приводить к перезагрузке СВТ без сохранения полученных результатов. Кроме того нужно учитывать, что невозможно использовать средства тестирования до загрузки ОС СЗИ или СВТ.

– Применение средств виртуализации для тестирования функций безопасности, не взаимодействующих с ОС СВТ, затруднено из-за невозможности перенаправления всех видов аппаратной компоненты в виртуальную среду. Также в данном случае неизвестна принципиальная возможность работы некоторых функций безопасности таких СЗИ в виртуальных машинах (ВМ). Например, перехват управления до загрузки ОС требует работы с прерываниями СВТ.

При использовании средств тестирования для функций безопасности программно-аппаратных СЗИ, не взаимодействующих с ОС СВТ и выполняющихся независимо от СВТ, также необходимо учитывать все рассмотренные выше особенности, только последняя из них видоизменяется: применение средств виртуализации в данном случае невозможно в принципе, так как такие СЗИ представляют собой отдельные модули, которые не могут взаимодействовать со средой виртуализации, кроме как по сети.

Для функций безопасности, взаимодействующих с ОС СВТ, применение средств тестирования не имеет первых двух описанных выше особенностей, так как СЗИ функционирует в той же среде, что и средства тестирования. В данном случае существует возможность использования отладчиков, дизассемблеров, а также средств автоматизации, фаззеров и так далее. Остается верной только особенность, связанная с применением средств виртуализации. В этом случае для USB-устройств перенаправление аппаратной компоненты в виртуальную среду возможно, но в процессе функционирования и тестирования аппаратной компоненты могут появляться новые ошибки, не возникающие на реальном СВТ и связанные с некорректной работой самих средств виртуализации.

Рассмотрим особенности применения средств тестирования функций безопасности программно-аппаратных СЗИ с точки зрения классификации по виду реализующей аппаратной компоненты (на базе стационарной и мобильной аппаратной компоненты). В данном случае существует особенность, которая связанна с возможностью отчуждения мобильной аппаратной компоненты, так как существует необходимость ее переподключения в процессе тестирования функций безопасности, используемых конечными пользователями. Программы тестирования должны учитывать отчуждаемость аппаратной компоненты и необходимость ее переподключения в ходе тестирования. Без применения дополнительных средств тестирования полноценно эмулировать работу реального пользователя таких СЗИ при автоматизации их тестирования невозможно. Для функций безопасности, реализованных на базе стационарных аппаратных компонент, таких особенностей в ходе тестирования нет, однако перед тестированием на определенном СВТ такую компоненту необходимо в него встраивать.

Таким образом, как и в случае анализа применимости существующих способов тестирования к программно-аппаратным СЗИ, особенности применения средств тестирования возникают при их рассмотрении с точки зрения двух первых признаков классификаций (по виду реализующей компоненты и виду взаимодействия с защищаемым СВТ в ИС). Наличие рассмотренных особенностей влияет на принципиальную возможность использования средств тестирования применительно к программно-аппаратным СЗИ. Для применения средств тестирования необходимо соответствующим образом изменять либо эти средства, либо СЗИ, в отношении которого они применяются. Для сертифицированных СЗИ последнее потребует дополнительной пересертификации. Кроме того, так как все рассмотренные средства представляют собой инструменты проведения тестирования, то необходимо разрабатывать программы тестирования функций безопасности программно-аппаратных СЗИ, которые бы использовали эти инструменты с учетом особенностей их применения. На основании этого можно сделать вывод, что требуется разработка средств тестирования для функций безопасности программно-аппаратных СЗИ.

Подводя итог, в настоящее время существует потребность в защите данных в информационных системах с применением программно-аппаратных средств защиты информации, для удовлетворения которой, в том числе необходимо проводить тестирование корректности функций безопасности этих средств и отсутствия негативного влияния на ИС.

Рассмотрев программно-аппаратные комплексы защиты информации как объекты тестирования, проведя классификацию их функций безопасности и проанализировав существующие способы и средства тестирования ПО применительно к выделенным классам, были определены характерные для каждого класса особенности, которые препятствуют их использованию по отношению ко всем функциям безопасности программно-аппаратных СЗИ.

При применении существующих способов тестирования программного обеспечения к функциям безопасности различных программно-аппаратных СЗИ могут возникать следующие особенности:

– сложность фиксации ошибок функционирования средств защиты (из-за ограниченности памяти СЗИ и/или перезагрузки СВТ при возникновении ошибок без сохранения результатов);

– необходимость тестирования большого количества различных исполнений аппаратных компонент СЗИ, имеющих отличающиеся входные условия функционирования и/или обладающих зависимостью от ОС СВТ;

– необходимость проведения перекрестного функционального тестирования составляющих, реализующих функции безопасности, при этом для некоторых СЗИ необходимо учитывать также зависимость от ОС СВТ;

– зависимость корректности или принципиальной возможности выполнения функций безопасности аппаратной компоненты СЗИ от аппаратной платформы, то есть необходимость проводить тестирование на разнообразных аппаратных платформах – с различными версиями BIOS/UEFI, различными прерываниями и питанием;

– сложность использования программного интерфейса (SDK) для тестирования аппаратной компоненты СЗИ (в некоторых случаях его целесообразно применять в среде ОС СЗИ, а не СВТ, что приводит к затруднению интерпретации результатов тестирования), при этом остается необходимость дополнительного тестирования совместно с программной компонентой;

– неоднозначность результатов нагрузочного, стрессового и функционального автоматизированного тестирования некоторых функций безопасности, реализованных в программной компоненте, запущенной на выполнение не в среде ОС аппаратного устройства, а в среде аналогичной установленной на СВТ ОС;

– сложность проведения проверок предельных значений характеристик аппаратной компоненты СЗИ, а в некоторых случаях – и программной, связанных с реализацией функций безопасности, вследствие необходимости выполнения этих проверок в ОС СЗИ в условиях ограниченной памяти аппаратной компоненты;

– необходимость переподключения отчуждаемой мобильной аппаратной компоненты в процессе тестирования используемых конечными пользователями функций безопасности, а также – переноса стационарной аппаратной компоненты из одного СВТ в другое для тестирования совместимости с различными аппаратными платформами.

Необходимо отметить, что некоторые из перечисленных особенностей, не имеющие зависимости от наличия в рассматриваемом средстве именно функций безопасности, характерны не только для СЗИ с аппаратной компонентой, но и для других программно-аппаратных комплексов.

При этом все выделенные особенности связаны с реализацией функций безопасности на аппаратном или программно-аппаратном уровне. Кроме того для программных и программно-аппаратных СЗИ в ходе тестирования могут быть выявлены ошибки, влияющие на безопасность системы, которые принципиально не могут существовать в ПО и других программно-аппаратных комплексах. Присутствие таких ошибок в СЗИ может потребовать другого подхода к проведению процесса верификации.

При применении существующих средств тестирования к функциям безопасности различных программно-аппаратных СЗИ могут возникать следующие особенности:

– сложность использования статических анализаторов и дизассемблеров – возможно реализовать путем переноса анализируемого исходного кода из ОС СЗИ в ОС СВТ с учетом того, что анализируемый код должен исполняться в условиях ограниченных ресурсов СЗИ;

– сложность или в ряде случаев невозможность использования фаззеров, отладчиков, средств автоматизированного тестирования и разработанных с их использованием программ тестирования, так как все эти средства также необходимо встраивать в ОС СЗИ, то есть может отсутствовать или существовать ограниченная возможность их применения вследствие недостаточного объема памяти СЗИ, а также возникновения критических ошибок в процессе функционирования средств тестирования без сохранения полученных результатов;

– ограниченная возможность применения средств виртуализации вследствие необходимости перенаправления аппаратной компоненты в виртуальную среду, при возможности такого перенаправления в процессе функционирования аппаратной компоненты могут появляться новые ошибки, связанные с некорректной работой средств виртуализации;

– возможность отчуждения мобильной аппаратной компоненты, а без применения вспомогательных средств тестирования полноценно эмулировать работу реального пользователя таких СЗИ при автоматизации их тестирования невозможно.

В широкоизвестных научных трудах в области тестирования большое внимание уделяется способам и средствам тестирования программного обеспечения. Необходимо отметить следующих авторов, которые внесли существенный вклад в создание основ, развитие теории и практики тестирования ПО: В. П. Котляров, И. В. Степанченко, Р. Блэк, Л. Тамре, С. Канер, С. В. Синицын и Н. Ю. Налютин – рассматривали основы и способы тестирования программного обеспечения, а двое последних также занимались изучением процесса верификации ПО; Б. Бейзер, в работах которого наиболее полно отражены вопросы тестирования черного ящика, а также функционального тестирования ПО; Э. Дастин, подробно рассматривающий особенности автоматизированного тестирования ПО; Р. Калбертсон и К. Бэк, работы которых посвящены изучению концепции экстремального программирования и быстрого тестирования ПО; В. В. Липаев и Г. Майерс, которые уделяли внимание проблемам качества и надежности программного обеспечения; В. И. Грекул, рассматривающий тестирование ПО в аспекте его внедрения в информационные системы.

Однако работы перечисленных авторов, в основном связаны с тестированием именно программного обеспечения, но недостаточное внимание уделено программно-аппаратным комплексам, в том числе программно-аппаратным СЗИ. В данных работах не учитываются особенности,

связанные, например, с тестированием функций безопасности или возникающие при реализации части функциональных возможностей на аппаратном уровне. Отличия в тестировании ПО и программно-аппаратных комплексов поверхностно рассмотрены в работах Р. Блэка и только на самом базовом уровне, при этом каких-либо методических рекомендаций по тестированию программно-аппаратных средств не приводится.

Вопросы верификации, как и вопросы тестирования, в известных научных работах и нормативных документах [15, 18, 24, 83, 95] рассматриваются только применительно к ПО, а также в рамках оценки надежности ИС и анализа рисков отказа программных средств. При этом обычно применяется вероятностный подход, в котором определяются качественные и количественные характеристики тяжести ошибок, появляющихся в ИС и приводящих к ее отказу с некоторой условной вероятностью и частотой [24, 95]. С одной стороны, такой подход можно использовать для предупреждения нарушений дальнейшей работы и построения более надежных систем, то есть в процессе поддержания качества ИС. С другой стороны, в данном случае проводится только управление рисками возможных ошибок, и не приводятся рекомендации по дальнейшим действиям при возникновении возможных ошибок во внедряемом ПО или средстве защиты. Кроме того, вероятностный подход предполагает, что в систему уже внедрено и используется ПО, относительно которого в дальнейшем и производится управление рисками. Поэтому такой подход не является превентивной мерой обеспечения надежности.

Таким образом, перечисленными авторами сформирована база для дальнейшего углубления и конкретизации теоретических и практических результатов одного из актуальных на данный момент направлений исследования – тестирования программно-аппаратных СЗИ.

В то же время вопросы тестирования программно-аппаратных СЗИ интересуют многих производителей, но применяемые ими на практике способы и средства тестирования не имеют теоретического описания, доступного в открытых источниках.

Наличие приведенных выше особенностей для программно-аппаратных СЗИ, а также отсутствие описаний способов и средств их тестирования в отрытых источниках обуславливает актуальность диссертационного исследования, а также формирует научную задачу работы, которая состоит в формировании модели и основанного на ней способа тестирования, а также рекомендаций по практической реализации средств тестирования, устанавливаемых в информационные системы программно-аппаратных средств защиты информации. Это позволит в условиях наличия особенностей и директивных сроков тестирования их функций безопасности обеспечить возможность проведения, полноту и оптимальность тестовых испытаний.

При этом объектом исследования являются программно-аппаратные СЗИ, функционирующие в среде ИС (ОС СВТ) или независимо от нее, и реализованные в них функции безопасности, а предметом исследования – способы и средства тестирования ПО применительно к функциям безопасности таких программно-аппаратных СЗИ.

1.5. Постановка задач исследования

В разделе 1.4 показана необходимость формирования нового научно-обоснованного способа тестирования для функций безопасности программно-аппаратных СЗИ. При этом из-за необхо-

димости организации регрессионного тестирования особое внимание требуется уделить именно автоматизации тестирования функций безопасности программно-аппаратных СЗИ, а также своевременной фиксации нарушений работы этих функций средой ИС, в которой используется данное средство защиты.

Разработку способа необходимо проводить с учетом особенностей тестирования функций безопасности программно-аппаратных СЗИ, которые возникают при их рассмотрении с точки зрения следующих признаков классификаций: по виду реализующей аппаратной компоненты (на базе стационарной аппаратной компоненты и на базе мобильной аппаратной компоненты) и виду взаимодействия с защищаемым СВТ в ИС (не взаимодействующие с ОС СВТ и взаимодействующие с этой средой).

Помимо этого при разработке нового способа тестирования программно-аппаратных СЗИ логично сначала сформулировать условия применимости существующих способов тестирования программного обеспечения к функциям безопасности программно-аппаратных СЗИ, определив отличия для различных видов функций безопасности. В случае наличия отличий, сформулировать условия применимости способов тестирования ПО для каждого вида функций безопасности программно-аппаратных СЗИ, имеющего свои особенности тестирования.

В дальнейшем с использованием полученных условий можно сформировать формальные критерии применимости существующих способов тестирования к функциям безопасности программно-аппаратных СЗИ. Для этого требуется разработать описательную модель программно-аппаратного СЗИ, а на ее основе построить математическую модель, например, с использованием аппарата теории формальных систем и теории автоматов. Полученные критерии применимости должны учитывать все выделенные в разделе 1.4 особенности, характерные для различных видов функций безопасности программно-аппаратных СЗИ. Кроме того, важно решить задачу обеспечения полноты и оптимальности тестирования, для чего модель СЗИ можно переформулировать уже с использованием теории графов и в последствии применить известные алгоритмы на графах для решения сформулированной задачи тестирования функций безопасности программно-аппаратных СЗИ.

Также необходимо выполнить разработку соответствующих алгоритмов тестирования и верификации функций безопасности программно-аппаратных СЗИ, которые позволили бы реализовать предлагаемый способ тестирования.

С использованием приведенных выше результатов становится возможным сформулировать и обосновать требования к средствам тестирования. Полученные требования должны учитывать выделенные в разделе 1.4 особенности, характерные для различных видов функций безопасности.

Для практической реализации средств тестирования и верификации необходимо сформулировать соответствующие рекомендации. С использованием данных рекомендаций, а также сформулированных требований, предложенных рекомендаций по способам проведения тестов, выявленных особенностей тестирования программно-аппаратных СЗИ на различных аппаратных платформах, рекомендаций по использованию средств виртуализации и по проведению процесса верификации СЗИ необходимо разработать программы тестирования функций безопасности

и требуемые технические или программно-аппаратные вспомогательные средства тестирования. Так как для каждого вида средств защиты они могут быть уникальными, то необходимо разработать программы тестирования и вспомогательные средства для программно-аппаратных СЗИ, реализующих различные виды функций безопасности.

В завершении необходимо провести анализ результатов применения разработанного способа и средств тестирования функций безопасности программно-аппаратных СЗИ, в том числе оценить ожидаемый эффект от их применения. Также необходимо описать апробацию, внедрение и реализацию полученных результатов работы. Дополнительно предполагается рассмотреть сторонние средства, облегчающие анализ результатов работы программ тестирования и делающие процесс тестирования более прозрачным, и средства, позволяющие тестировать компоненты СЗИ, не реализующие функции безопасности напрямую, и возможность их интеграции в комплекс тестирования программно-аппаратных СЗИ.

Таким образом, можно сформулировать цель исследования: разработка научно-обоснованного способа тестирования функций безопасности программно-аппаратных средств защиты информации.

При этом для реализации данной цели, как обосновано в данном разделе выше, необходимо решить следующие задачи:

1. Анализ известных подходов, используемых для тестирования программного обеспечения и обоснование их ограниченности в задаче полной проверки функций безопасности программно-аппаратных СЗИ.

2. Разработка научно-обоснованного способа тестирования функций безопасности программно-аппаратных СЗИ, основанного на использовании модели программно-аппаратных СЗИ и соответствующих алгоритмов тестирования и верификации их функций безопасности.

3. Разработка программно-аппаратного комплекса, реализующего предложенный способ тестирования функций безопасности программно-аппаратных СЗИ.

4. Апробация и анализ результатов применения разработанного способа тестирования и программно-аппаратного комплекса для тестирования функций безопасности программно-аппаратных СЗИ.

Выводы

В результате проведенного в данной главе анализа получены следующие результаты:

1. Показано, что тестирование встраиваемых в ИС программно-аппаратных СЗИ необходимо проводить для проверки корректности функционирования всех реализованных функций безопасности и отсутствия негативного влияния средства защиты на характеристики системы. А для своевременной фиксации возможных ошибок необходимо автоматизировать данный процесс.

2. Выявлены виды программно-аппаратных СЗИ и получена классификационная схема их функций безопасности, согласно которой они разделяются: по виду реализующей аппаратной компоненты (на базе стационарной или мобильной); по виду взаимодействия с защищаемым

СВТ в ИС (не взаимодействующие с ОС СВТ или взаимодействующие с ее средой); по виду угроз, которым противодействует (от угроз конфиденциальности, от угроз целостности или от угроз доступности) и типу противодействия (превентивные или пресекающие/противодействующие).

3. Определены особенности тестирования, препятствующие возможности применения существующих способов тестирования ПО по отношению к различным видам функций безопасности программно-аппаратных СЗИ, с точки зрения вида реализующей их аппаратной компоненты и вида ее взаимодействия с защищаемым СВТ. Данные особенности характерны исключительно для программно-аппаратных СЗИ, а не для ПО или программных СЗИ.

4. Приведено обоснование ограниченности известных подходов, используемых для тестирования программного обеспечения, в задаче полной проверки функций безопасности аппаратно-программных СЗИ. Показана необходимости модификации данных подходов, а также соответствующих способов и средств тестирования.

5. Сформулирована цель диссертационного исследования и его задачи, которые необходимо решить для достижения данной цели.

2. Разработка научно-обоснованного способа тестирования функций безопасности программно-аппаратных СЗИ, основанного на использовании модели программно-аппаратных СЗИ

В данной главе выполняется разработка описательной и формальной модели программно-аппаратного СЗИ. Предлагаются условия применимости способов тестирования ПО к различным видам функций безопасности программно-аппаратных СЗИ, учитывающие особенности их тестирования. С использованием данных условий для предложенной формальной модели разрабатываются критерии применимости существующих способов тестирования к функциям безопасности программно-аппаратных СЗИ, а также частные критерии для различных видов этих функций безопасности. Исследуется задача обеспечиения полноты и оптимальности тестирования программно-аппаратных СЗИ. Помимо этого исследуется процесс верификации программно-аппаратных СЗИ и оценка влияния выявленных в ходе тестирования функций безопасности ошибок на информационную систему, в которой они используются. Предложен способ тестирования функций безопасности программно-аппаратных СЗИ, учитывающий состояния аппаратной компоненты, применимый для различных видов программно-аппаратных средств защиты информации.

2.1. Разработка модели программно-аппаратных средств защиты информации, реализующих подлежащие тестированию функции безопасности

На основании перечня функций безопасности программно-аппаратных СЗИ, построенного в разделе 1.2, определим описательную модель таких средств защиты, реализующих подлежащие тестированию функции безопасности.

Из представленного в разделе 1.2 перечисления видно, что существует большое количество функций безопасности, реализующих различную функциональность обеспечения информационной безопасности ИС, каждую из которых необходимо тестировать. При этом необходимо учитывать, что некоторые функции безопасности могут дублироваться в различных программно-аппаратных СЗИ одного производителя. Например, функции идентификации и аутентификации – в средствах обеспечения доверенной загрузки ОС и разграничения доступа пользователей в ОС; функция контроля целостности – в средствах обеспечения доверенной загрузки ОС, разграничения доступа пользователей в ОС и защищенной загрузки ПО терминальных станций по сети; функция шифрования – в средствах криптографической защиты информации и служебных носителях информации, функции вычисления и проверки подписи – в средствах криптографической защиты информации, обеспечения доверенного сеанса связи и защищенной загрузки ПО терминальных станций по сети и так далее. Однако такое дублирование не означает, что достаточно протестировать только одну из этих функций и на этом основании будет подтверждена работоспособность другой, использующей тот же программный код при реализации.

Кроме того, для выполнения некоторых функций безопасности программно-аппаратных СЗИ необходимо предварительно завершить другие, иначе эти функции не могут быть выполнены в принципе. Это характерно не только для СЗИ, но и для других средств. У средств защиты,

как правило, существует связь между различными функциями безопасности, реализованными в одном программно-аппаратном средстве защиты. Например, для шифрования необходимо сначала сгенерировать соответствующие симметричные ключи; для вычисления и проверки подписи – сгенерировать ключевые пары; для разграничения доступа в ОС – пройти идентификацию и аутентификацию пользователя и так далее.

Также для всех возможных средств защиты существует еще одна зависимость возможности выполнения различных функций безопасности, которую нужно учитывать при тестировании – состояние реализующего их СЗИ. При этом для программных СЗИ – это состояние программной компоненты, а для программно-аппаратных – и программной, и аппаратной компоненты (в общем случае не зависящих, но влияющих друг на друга). Например, такая функция безопасности, как шифрование, может быть выполнена только, если программная компонента СЗИ находится в следующем состоянии – подготовлены данные для шифрования, например, переданы от какого-либо внешнего ПО в СЗИ. А аппаратная компонента инициализирована, сгенерированы ключи шифрования, успешно пройдена идентификация и аутентификация пользователя. Если хотя бы одна из компонент в необходимый момент времени не будет находится в нужном состоянии, то функция безопасности программно-аппаратного СЗИ не будет выполнена. Для каждого программно-аппаратного СЗИ существует достаточно большое число состояний программной и аппаратной компонент, количество которых с учетом их сочетания может быть пропорционально количеству функций безопасности, то есть в разы больше чем для программных СЗИ.

Как правило, для всех программно-аппаратных СЗИ существуют следующие общие состояния аппаратной компоненты: неинициализирована (не отформатирована, не заданы параметры авторизации, пароль администратора и пользователя); инициализирована (отформатирована, заданы параметры авторизации, пароль администратора и пользователя); подготовлена к работе (например, сгенерированы или импортированы криптографические ключи и сертификаты, необходимые для выполнения процедур шифрования и подписи, или зарегистрирован пользователь для осуществления защищенного входа в ОС, или существует одновременное сочетание двух или более перечисленных пунктов); технологический режим (подготовлена для перезаписи внутреннего ПО). Для программно-аппаратных СЗИ, функции безопасности которых реализованы в мобильной аппаратной компоненте (см. разделы 1.2–1.4) также существует еще два состояния: аппаратная компонента подключена к СВТ и не подключена к нему. Смена между указанными состояниями осуществляется многократно в процессе работы средства защиты. А для реализованных в стационарной аппаратной компоненте – встроена в СВТ и отключена от него. Смена между указанными состояниями осуществляется перед началом работы с СЗИ и, в некоторых случаях, после окончания работы с ним. При этом две пары последних перечисленных состояний могут сочетаться с описанными выше общими для всех программно-аппаратных СЗИ состояниями, что создает еще больше возможных вариантов по сравнению с программными средствами защиты.

Программная же компонента программно-аппаратных СЗИ имеет два общих состояния, характерных для всех таких средств защиты: установлена и не установлена в ОС СВТ. В средствах защиты, обладающих собственной операционной системой, программная компонента, как пра-

вило, предустановлена в этой ОС. Остальные состояния специфичны для конкретных функций безопасности, которые реализует рассматриваемое программно-аппаратное СЗИ.

Переход из одного состояния программной или аппаратной компоненты в другое можно осуществить путем выполнения функций безопасности, нецелевых функций СЗИ или с помощью выполнения каких-либо внешних (физических) воздействий. Примерами нецелевых функций СЗИ являются инициализация аппаратной компоненты, настройка программной компоненты и так далее. А примерами внешних воздействий – замыкание контактов аппаратной компоненты путем перевода переключателя в нужное положение, подключение к СВТ и так далее.

Тестирование функций безопасности программно-аппаратных СЗИ, для которых требуется переход в какое-либо состояние путем выполнения физических действий, часто возможно только ручным способом (см. раздел 1.3), так как автоматизация этих действий с использованием средств автоматизированного тестирования бывает невозможна или затруднена. Для других функций безопасности применение автоматизированного и автоматического способов тестирования возможно при учете других специфичных только для них особенностей СЗИ.

Формализуем полученную описательную модель программно-аппаратных СЗИ, реализующих функции безопасности, подлежащие тестированию. Для этого воспользуемся аппаратом системного анализа, теории формальных систем, множеств и автоматов.

Обозначим как M – множество [3] всех программно-аппаратных СЗИ. Обозначим через M_p множество программно-аппаратных СЗИ, для которых вне зависимости от человеческого фактора возможно выполнение ручного тестирования (с применением способа ручного тестирования).

В общем случае $M_p \subseteq M$, то есть любые непротиворечивые программно-аппаратные СЗИ могут быть проверены вручную. Однако непротиворечивость конкретного СЗИ, то есть существование принципиальной возможности реализации всех его функций безопасности, необходимо обосновывать.

В множество M также входят следующие подмножества [29, 104]:

$$M_a \subseteq M_{na} \subseteq M,$$

где M_a – множество программно-аппаратных СЗИ, тестирование которых можно выполнить автоматически (с применением способа автоматического тестирования). M_{na} – множество программно-аппаратных СЗИ, тестирование которых возможно автоматизировать не полностью, а частично. В данном случае необходимо применение способа автоматизированного тестирования, при котором, как правило, большинство проверок СЗИ можно проводить полностью автоматически, но некоторые действия, необходимые для выполнения одной из проверок можно провести только вручную, например, переподключение Аккорд-АМДЗ в PCI-слот другого СВТ.

При этом: $M_a \subseteq M_p$, $M_{na} \subseteq M_p$ – подмножества множества M_p. Множество программно-аппаратных СЗИ (M) представлено на Рисунке 2.1.

Также множество M может быть представлено следующим образом [29, 104]:

$$M = P_{oc} \cup P_{noc},$$

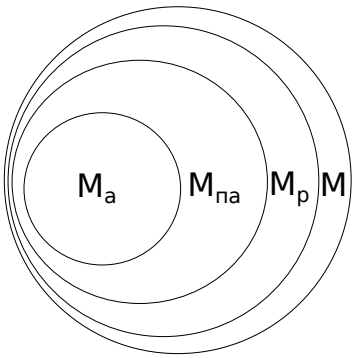


Рисунок 2.1 – Множество программно-аппаратных СЗИ

где $P_{ос}$ – множество программно-аппаратных СЗИ, функции безопасности которых взаимодействуют со средой ОС СВТ; $P_{нос}$ – множество программно-аппаратных СЗИ, функции безопасности которых не взаимодействуют с ОС СВТ.

Множество $P_{нос}$ состоит из двух подмножеств: $P_{нос} = P_{встр} \cup P_{незав}$,

где $P_{встр}$ – множество встраиваемых в СВТ программно-аппаратных СЗИ, функции безопасности которых выполняются независимо от ОС в составе СВТ; $P_{незав}$ – множество программно-аппаратных СЗИ, функции безопасности которых выполняются независимо от СВТ.

При этом $P_{встр} \cap P_{незав} = \varnothing$.

Помимо этого множество M может быть представлено в виде объединения следующих подмножеств: $M = P_{моб} \cup P_{стац} \cup P_{незав}$,

где $P_{моб}$ – множество программно-аппаратных СЗИ, функции безопасности которых реализованы на базе мобильной аппаратной компоненты; $P_{стац}$ – множество программно-аппаратных СЗИ, функции безопасности которых реализованы на базе стационарной аппаратной компоненты.

При этом $P_{моб} \cap P_{стац} = \varnothing$, $P_{моб} \cap P_{незав} = \varnothing$ и $P_{стац} \cap P_{незав} = \varnothing$.

$P_{моб} \cup P_{стац} = P_{ос} \cup P_{встр}$ – разные представления множеств одних и тех же программно-аппаратных СЗИ.

Зафиксируем произвольное $m \in M$, которое в дальнейшем будем рассматривать как общий случай программно-аппаратного СЗИ и для которого сформулируем соответствующие определения и критерии.

Обозначим $V = \{v_0, v_1, \ldots, v_n\}$, $n \in \mathbb{N}_0$ – множество состояний, в которых может находиться m (различные сочетания состояний программной и аппаратной компонент, как описано выше). При этом $v_0 \in V$ – начальное состояние (например, когда программная компонента не установлена и/или аппаратная – не инициализирована).

Обозначим $I = I_{фб} \cup I_{нц} \cup I_{вн}$ – множество функций m, которые могут выполняться в состояниях $v \in V$, и являются либо функциями безопасности с формальными параметрами ($I_{фб}$, например, идентификация, аутентификация, шифрование, подпись и контроль целостности), либо нецелевыми функциями программно-аппаратного СЗИ ($I_{нц}$), либо внешними воздействиями на него ($I_{вн}$, например, ввод PIN-кода, переподключение аппаратной компоненты). $I_{фб} \cap I_{нц} = \varnothing$, $I_{фб} \cap I_{вн} = \varnothing$, $I_{нц} \cap I_{вн} = \varnothing$.

При этом $I_{фб}$ – множество функций безопасности программно-аппаратного СЗИ m, которые нужно протестировать.

В общем случае для любого $m \in M$ выполнение функции безопасности $i \in I_{фб}$ должно проверяться в одном или нескольких состояниях из V. В случае проверки нескольких параметров для определенной функции безопасности, во множество V должны быть включены состояния, соответствующие всем возможным сочетаниям параметров для данной функции безопасности. Таким образом, тестирование различных параметров функций безопасности будет состоять в обходе таких состояний программно-аппаратного СЗИ.

Как было сказано ранее, некоторые функции безопасности программно-аппаратного СЗИ могут быть выполнены только в определенных состояниях, в которых находится программная и/или аппаратная компоненты средства защиты. Например, нельзя выполнять никакую функцию СЗИ до того, как установлено программное обеспечение или, как правило, до того, как выполнится первоначальная инициализация его аппаратной компоненты, или нельзя использовать функции шифрования СКЗИ, если не сгенерированы ключи шифрования. При этом выполнимость или возможность проверки функции безопасности в определенном состоянии подразумевает наличие вычисляющего ее алгоритма. Если функция безопасности невыполнима в определенном состоянии, то ее тестирование в этом состоянии провести невозможно [29, 104] и не требуется при решении задачи тестирования в соответствии с построением множества V. Основываясь на этом, введем следующее определение.

Будем называть $v \in V$ – состоянием с потенциально вычислимыми функциями безопасности, если существуют функции безопасности, которые необходимо проверить в состоянии v. Обозначим $V_{фб} \subseteq V$ – множество всех состояний программно-аппаратного СЗИ с потенциально вычислимыми функциями безопасности – состояний, в которых необходимо проверить функции безопасности m.

Построим математическую модель [10, 25, 51] некоторого абстрактного программно-аппаратного СЗИ $m \in M$ в виде конечного детерминированного автомата: $\widetilde{m} = (V, I, O, f, g)$ [106], для которого:

– V – множество состояний автомата;

– v_0 – начальное состояние автомата, в котором находится m в начале тестирования;

– I – множество входов (стимулов) автомата – функций m, которые могут выполняться в состояниях $v \in V$;

– $O = \{0, 1\}$ – множество выходов (реакций) – результатов выполнения стимулов в состояниях $v \in V$ (успешное или неуспешное выполнение);

– $f : I \times V \to V$ – функция переходов автомата: если $f((i,v)) = v'$, то по стимулу $i \in I$ из состояния $v \in V$ автомат переходит в состояние $v' \in V$.

– $g : I \times V \to O$ – функция выходов автомата: если $g((i,v)) = o$, то по стимулу $i \in I$ из состояния $v \in V$ на выход автомата поступает $o \in O$.

Обозначим $T \subseteq V \times I \times O \times V$ – множество всевозможных переходов автомата, при каждом переходе (v, i, o, v') выполняется $f((i,v)) = v'$ и $g((i,v)) = o$.

Определение 1. Последовательностями переходов тестирования для программно-аппаратного СЗИ m, моделируемого с помощью автомата $\widetilde{m} = (V, I, O, f, g)$, называются пустая последовательность $\overline{s}_0$ длины $len(\overline{s}_0) = 0$, а также последовательности $\overline{s}$ переходов $s_0, \ldots, s_{l-1} \in T$ длины $len(\overline{s}) = l \in \mathbb{N}$. При этом для $\forall l \in \mathbb{N}$ элементы последовательности $\overline{s}$ имеют вид: $s_j = (v'_j, i'_{j+1}, o'_{j+1}, v'_{j+1}) \in T$, где $j = 0, \ldots, l-1$, а для первого элемента последовательности $s_0 = (v'_0, i'_1, o'_1, v'_1) \in T$ выполняется $v'_0 = v_0$.

Каждая возможная последовательность переходов тестирования начинается из начального состояния v_0, может обходить произвольное количество состояний $v \in V$ (элементы последовательности могут повторяться) и имеет конечную длину. В последовательности $\overline{s}_0$ не осуществляется ни одного перехода из v_0.

Обозначим множество всевозможных последовательностей переходов тестирования различной длины S.

Задача тестирования функций безопасности программно-аппаратного СЗИ m заключается в обходе состояний автомата $\widetilde{m} = (V, I, O, f, g)$ с помощью последовательности переходов тестирования, при этом:

– для обеспечения полноты тестирования в каждое состояние $v \in V_{фб} \subseteq V$ необходимо перейти с помощью всех возможных стимулов $i \in I$ (с разным набором параметров и входных условий);

– для обеспечения оптимальности тестирования длина последовательности переходов тестирования, обеспечивающей полноту, должна быть минимальна.

Определение 2. Будем говорить что для программно-аппаратного СЗИ m, моделируемого автоматом $\widetilde{m} = (V, I, O, f, g)$, решена задача тестирования с помощью последовательности переходов тестирования $\overline{s} \in S$ из Определения 1, когда одновременно выполняются приведенные ниже условия [106]:

– условие возможности проведения тестирования:

$$m \in M_p \tag{2.1}$$

– условие полноты тестирования:

$$\forall s_j \in T, \text{ для которого } v'_{j+1} \in V_{фб}, \text{ является элементом последовательности } \overline{s} \tag{2.2}$$

– условие оптимальности тестирования:

$$len(\overline{s}) = \min\{len(\overline{s}') : \forall \overline{s}' \in S \text{ выполняется (2.2)}\} \tag{2.3}$$

2.2. Доказательство ограниченности способов, используемых для тестирования программного обеспечения, в задаче полной проверки функций безопасности программно-аппаратных средств защиты информации

Решение задачи тестирования из Определения 2 для конкретного программно-аппаратного СЗИ в общем случае может существовать не всегда. При этом чаще всего может нарушаться условие (2.1) о принципиальной возможности проведения тестирования, то есть возможности проверить необходимые функции безопасности во множестве состояний с потенциально вычислимыми функциями безопасности. В данном разделе сформулируем условия применимости существующих способов тестирования программного обеспечения к функциям безопасности программно-аппаратных СЗИ, используя выявленные в разделе 1.4 особенности их тестирования. Так как данные особенности зависят от вида функций безопасности, то формулировать условия применимости необходимо для каждого из этих видов. На основании полученных условий с использованием математической модели программно-аппаратных СЗИ (см. раздел 2.1), а также ключевых положений системного анализа, теорий формальных систем и автоматов станет возможным формализовать критерии применимости существующих способов тестирования программного обеспечения к функциям безопасности программно-аппаратных СЗИ, что позволит уже на первом этапе определить возможно ли решить задачу тестирования из Определения 2 для конкретного программно-аппаратного СЗИ.

2.2.1. Условия применимости существующих способов тестирования программного обеспечения для тестирования функций безопасности программно-аппаратных средств защиты информации

Проанализируем особенности тестирования программно-аппаратных СЗИ, реализующих различные виды функций безопасности, из раздела 1.4 с точки зрения возможности применения способов ручного, автоматизированного и автоматического тестирования.

В отношении тестирования функций безопасности программно-аппаратных СЗИ, не взаимодействующих с ОС СВТ и выполняющихся независимо от ОС в составе СВТ, характерны следующие особенности:

1. Зависящие от среды, в которой выполняются функции безопасности (ОС СЗИ), и затрудняющие применение существующих способов и средств тестирования:

– сложность фиксации ошибок функционирования средств защиты – при применении способа ручного тестирования фиксацию ошибок возможно выполнить либо визуально, либо, как для способов автоматизированного и автоматического тестирования, путем журналирования в памяти СЗИ с возможным переполнением этой памяти или потерей информации о некоторых событиях;

– сложность использования программного интерфейса для тестирования аппаратной компоненты СЗИ – способы автоматизированного и автоматического тестирования целесообразно применять в среде ОС СЗИ, а не СВТ, что приводит к затруднению интерпретации результатов тестирования;

– неоднозначность результатов нагрузочного, стрессового и функционального автоматизированного или автоматического тестирования некоторых функций безопасности, реализованных в программной компоненте, запущенной на выполнение не в среде ОС аппаратного устройства, а в среде аналогичной ОС, установленной на СВТ;

– сложность проведения ручных или автоматизированных проверок предельных значений характеристик аппаратной и программной компонент СЗИ, связанных с реализацией функций безопасности, вследствие необходимости выполнения этих проверок в ОС СЗИ;

– сложность использования статических анализаторов и дизассемблеров при применении любого из существующих способов тестирования;

– сложность или в ряде случаев невозможность использования в любом из существующих способов тестирования фаззеров, отладчиков, средств автоматизированного тестирования и разработанных с их использованием программ тестирования, так как все эти средства также необходимо встраивать в ОС СЗИ;

– ограниченная возможность применения средств виртуализации в любом из существующих способов тестирования вследствие необходимости перенаправления аппаратной компоненты в виртуальную среду, а также зависимости некоторых функций безопасности от физических СВТ.

2. Многократно увеличивающие количество проводимых проверок и объем тестирования:

– необходимость тестирования большого количества различных исполнений аппаратных компонент СЗИ, имеющих отличающиеся входные условия функционирования и другие параметры – в этом случае объемы проверок функций безопасности увеличиваются пропорционально количеству используемых исполнений (для каждого из которых часть проверок может быть уникальной), что значительно усложняет применение способа ручного тестирования и требует использования автоматизированного или автоматического тестирования;

– необходимость проведения перекрестного функционального тестирования составляющих, реализующих функции безопасности – проведение такого тестирования вручную также затруднено (аналогично пункту выше);

– зависимость корректности или принципиальной возможности выполнения функций безопасности аппаратной компоненты СЗИ от аппаратной платформы, то есть существует необходимость проводить тестирование на разнообразных аппаратных платформах с различными версиями BIOS/UEFI, прерываниями и другими особенностями – в данном случае возможно применение способа ручного тестирования, но применение способа автоматизированного и автоматического тестирования достаточно затруднено.

В соответствии с вышесказанным применение известных способов тестирования для функций безопасности программно-аппаратных СЗИ, выполняющихся независимо от ОС в составе СВТ, возможно при выполнении следующих условий [29, 104].

Условие 1. Существуют необходимые условия для:

– функционирования аппаратной компоненты, реализующей функции безопасности программно-аппаратного СЗИ (для физического СВТ – корректное функционирование в составе

аппаратной платформы, для среды виртуализации – возможность перенаправления аппаратной компоненты и корректное функционирование в составе виртуальной машины);

– работы среды функционирования средств тестирования (статических анализаторов, дизассемблеров, фаззеров и отладчиков, а для автоматизированного и автоматического тестирования еще и средств автоматизации), а также используемого программного интерфейса (работоспособность в среде ОС, из которой проводится тестирование).

Условие 2. Присутствует возможность фиксации результатов тестирования функций безопасности программно-аппаратного СЗИ.

Для автоматизированного и автоматического тестирования необходимо выполнение дополнительного условия.

Условие 3. Существуют и применяются программы тестирования функций безопасности, в том числе для нагрузочного, стрессового и перекрестного функционального тестирования компонент программно-аппаратного СЗИ.

Из полученных условий видно, что для тестирования (вне зависимости от выбранного способа) может потребоваться использование некоторого опытного образца аппаратной компоненты. Этот образец может быть доработкой штатного исполнения аппаратной компоненты, например, путем увеличения объема памяти или добавления внешнего носителя информации для хранения результатов тестирования, а также других изменений схемотехники. Такая доработка скорее всего также потребует доработки программной компоненты. Однако тестирование такого доработанного СЗИ не гарантирует точно такой же результат для его стандартного исполнения. Помимо этого необходимо учитывать, что при использовании средств виртуализации в процессе тестирования аппаратной компоненты могут появляться ошибки, связанные с некорректной работой самих этих средств.

Таким образом, для функций безопасности программно-аппаратных СЗИ, выполняющихся независимо от ОС в составе СВТ, можно использовать способы ручного и автоматизированного тестирования, но для этого потребуется изменять как программную, так и аппаратную компоненту СЗИ. Применение способа автоматического тестирования затрудняется необходимостью проведения проверок работоспособности СЗИ на различных СВТ, а также – функций безопасности, выполняющихся до загрузки ОС СВТ или СЗИ.

В процессе тестирования функций безопасности программно-аппаратных СЗИ, не взаимодействующих с ОС СВТ и выполняющихся независимо от СВТ, также возникают перечисленные выше особенности, кроме двух – нет зависимости от аппаратной платформы, а также отсутствует принципиальная возможность применения средств виртуализации. Такие СЗИ могут взаимодействовать со средой виртуализации только по сети. Поэтому для возможности применения способов ручного и автоматизированного тестирования к функциям безопасности программно-аппаратных СЗИ этого вида должны также выполняться Условия 1-2 и Условия 1-3 соответственно (за исключением первой части Условия 1). При этом из-за отсутствия зависимости от аппаратной платформы в случае выполнения данных Условий 1-3 не возникает сложностей при применении способа автоматического тестирования.

Для функций безопасности, взаимодействующих с ОС СВТ, существует возможность фиксации ошибок и программной и аппаратной компоненты в операционной системе СВТ. Кроме того, для СЗИ с аппаратной компонентой на базе USB-устройств, нет необходимости проведения тестирования на разнообразных аппаратных платформах, достаточно соответствия спецификации определенной версии интерфейса USB. Также для нее по умолчанию возможно перенаправление в виртуальную среду для большинства средств виртуализации. Помимо этого вследствие отсутствия ОС СЗИ, функции безопасности рассматриваемых средств защиты изначально тестируются в среде операционной системы (а не в среде СЗИ) и при этом возможно использование отладчиков, дизассемблеров и других средств тестирования – средств автоматизации, фаззеров и так далее. В общем случае для возможности применения способов ручного и автоматизированного тестирования к функциям безопасности программно-аппаратных СЗИ этого вида должны также выполняться Условия 1-2 и Условия 1-3 соответственно. При этом Условия 1-2 в большинстве случаев могут выполняться по умолчанию и не требуют выполнения серьезных доработок, как в случае с функциями безопасности, выполняющимися независимо от ОС в составе СВТ. В данном случае при выполнении Условий 1-3 при применении способа автоматического тестирования не возникает каких-либо препятствий.

Рассмотрим особенности, характерные для функций безопасности программно-аппаратных СЗИ, разделяемых по виду реализующей аппаратной компоненты. Для функций безопасности, реализованных на базе мобильной аппаратной компоненты, такой особенностью является возможность отчуждения этой компоненты в процессе работы СЗИ. Поэтому существует необходимость переподключения аппаратной компоненты средства защиты в процессе применения (а, значит, и тестирования) функций безопасности, используемых конечными пользователями. В соответствии с вышесказанным применение существующих способов тестирования для функций безопасности программно-аппаратных СЗИ, реализованных на базе мобильной аппаратной компоненты, возможно при выполнении дополнительного условия [29, 104].

Условие 4. Существует возможность подключения/переподключения в процессе настройки или использования мобильной аппаратной компоненты, реализующей функции безопасности программно-аппаратного СЗИ.

При ручном тестировании данное условие может быть выполнено, например, тестировщиком путем физического подключения и отключения аппаратной компоненты от СВТ. Однако при автоматизированном и автоматическом тестировании необходимо применение вспомогательных средств тестирования, позволяющих эмулировать работу реального пользователя, в том числе, в случае необходимости подключения/переподключения СЗИ в ходе настройки или использования [34].

При тестировании функций безопасности, реализованных на базе стационарной аппаратной компоненты, особенности, связанной с ее отчуждением, не возникает. Однако, как правило, существует необходимость переноса стационарной аппаратной компоненты в другие СВТ для проверки взаимодействия с различными аппаратными платформами. В соответствии с этим применение существующих способов тестирования для функций безопасности программно-

аппаратных СЗИ, реализованных на базе стационарной аппаратной компоненты, возможно при выполнении следующего условия [29, 104].

Условие 5. Существует возможность переноса стационарной аппаратной компоненты, реализующей функции безопасности программно-аппаратного СЗИ, из одного СВТ в другое для тестирования на различных аппаратных платформах.

При ручном тестировании данное условие может быть выполнено путем физического распломбирования корпуса СВТ, откручивания крепежных болтов, снятия крепежей, извлечения контроллера, переноса его в другое СВТ, а затем выполнения тех же действий в другом СВТ в обратном порядке. Применение способов автоматизированного и автоматического тестирования в данном случае возможно только при использовании некоторых вспомогательных средств тестирования, позволяющих эмулировать эти действия.

Помимо выполнения Условий 4 и 5, при применении существующих способов тестирования для функций безопасности программно-аппаратных СЗИ, реализованных на базе мобильной и стационарной аппаратной компоненты, в зависимости от вида взаимодействия рассматриваемых функций безопасности с защищаемым СВТ в ИС, необходимо учитывать также соответствующую комбинацию Условий 1-3 аналогично тому, как это было рассмотрено выше.

Таким образом, предложены условия применимости способов тестирования ПО (программных СЗИ) к различным видам функций безопасности программно-аппаратных СЗИ. На основе полученных условий можно формализовать критерии применимости способов тестирования ПО к функциям безопасности программно-аппаратных СЗИ.

2.2.2. Критерии применимости существующих способов тестирования для тестирования функций безопасности программно-аппаратных средств защиты информации

Формализуем критерии применимости способов тестирования ПО к функциям безопасности программно-аппаратных СЗИ [29] с использованием математической модели программно-аппаратных СЗИ из раздела 2.1, а также ключевых положений системного анализа, теорий формальных систем и автоматов.

Для выполнения условия (2.1) при решении задачи тестирования программно-аппаратного СЗИ из Определения 2 необходимо удостовериться в принципиальной возможности провести тестирование, то есть в принадлежности конкретного $m \in M$ множествам M_p, M_a или $M_{па}$. В рамках модели программно-аппаратного СЗИ $\widetilde{m} = (V, I, O, f, g)$ такая принадлежность означает, что существует возможность проверить необходимые функции безопасности во множестве состояний с потенциально вычислимыми функциями безопасности $V_{фб} \subseteq V$. Однако, изначально программно-аппаратное СЗИ находится в состоянии $v_0 \in V$, и в состояния из $V_{фб}$ нужно попасть с помощью переходов из некоторой последовательности переходов тестирования $\overline{s} \in S$ длины $len(\overline{s}) = l \in \mathbb{N}_0$. При этом, при осуществлении переходов $s_j \in T, j = 0, \ldots, l-1$ и обходе состояний из $V_{фб}$ должны выполняться соответствующие необходимые Условия 1–5 из раздела 2.2.1.

Определение 3. Функция $i \in I_{фб}$ называется вычислимой функцией безопасности в некотором состоянии $v \in V_{фб}$ автомата $\widetilde{m} = (V, I, O, f, g)$, если для нее в этом состоянии выполняются Условия 1–2 из раздела 2.2.1.

Определение 4. Функция $i \in I_{нц} \cup I_{вн}$ называется вычислимым стимулом в некотором состоянии $v \in V$ автомата $\widetilde{m} = (V, I, O, f, g)$, если для нее в этом состоянии выполняются соответственно Условия 1–2 или 4–5 из раздела 2.2.1.

Пусть автомат $\widetilde{m} = (V, I, O, f, g)$ находится в состоянии v_0. В этом случае обеспечивается возможность проведения ручного тестирования функций безопасности программно-аппаратных СЗИ и для моделируемого этим автоматом выполняется $m \in M_p$ тогда и только тогда, когда существует последовательность переходов тестирования $\overline{s} \in S$ длины $len(\overline{s}) = l \in \mathbb{N}_0$ и для $\forall i \in I_{фб}$ выполняется одно из следующих условий:

1. i является вычислимой функцией безопасности (в соответствии с Определением 3) в состоянии v_0;

2. i является вычислимой функцией безопасности (в соответствии с Определением 3) в состоянии $v'_j \in V_{фб}$, $1 \leqslant j \leqslant l$ и $\overline{s}$ содержит последовательные переходы $s_0, \ldots, s_{j-1} \in T$ с началом в $v'_0 = v_0$, такие, что $i'_1, \ldots, i'_j$ являются вычислимыми стимулами (в соответствии с Определением 4) в состояниях $v'_0, \ldots, v'_{j-1}$ соответственно.

Предположим, что $m \in M_p$, то есть программно-аппаратное СЗИ может быть проверено вручную. Автомат, соответствующий данному СЗИ, $\widetilde{m} = (V, I, O, f, g)$ находится в начальном состоянии $v_0 \in V$. Так как программно-аппаратное СЗИ может быть проверено вручную, то это означает что $\forall i \in I_{фб}$ можно проверить вручную, то есть они являются вычислимыми функциями безопасности (в соответствии с Определением 3) в некоторых состояниях из $V_{фб}$ и эти состояния достижимы из v_0. То есть либо все i – вычислимые функции безопасности (в соответствии с Определением 3) в v_0, либо для тех i, которые не являются вычислимыми функциями безопасности в v_0, существуют вычислимые стимулы (в соответствии с Определением 4), которые позволяют перейти в соответствующие $v'_j \in V_{фб}$, в которых они являются вычислимыми функциями безопасности.

Если же $\forall i \in I_{фб}$ – вычислимые функции безопасности в v_0, то очевидно, что все множество функций безопасности можно проверить вручную в состоянии v_0.

В противном случае очевидно, что из v_0 достижимы состояния $v'_j \in V_{фб}$, в которых вычислимы $\forall i \in I_{фб}$, то есть все множество функций безопасности можно проверить вручную в таких состояниях v'_j.

Определение 5. Функция $i \in I_{фб}$ называется автоматически вычислимой функцией безопасности в некотором состоянии $v \in V$ автомата $\widetilde{m} = (V, I, O, f, g)$, если для нее в этом состоянии выполняются Условия 1–3 из раздела 2.2.1.

Определение 6. Функция $i \in I_{нц} \cup I_{вн}$ называется автоматически вычислимым стимулом в некотором состоянии $v \in V$ автомата $\widetilde{m} = (V, I, O, f, g)$, если для нее в этом состоянии выполняются соответственно Условия 1–3 или 3–5 из раздела 2.2.1.

В соответствии с этим формализуем общие критерии возможности автоматического и автоматизированного тестирования функций безопасности программно-аппаратных СЗИ.

Общий критерий возможности автоматического тестирования функций безопасности программно-аппаратных СЗИ:

Пусть автомат $\widetilde{m} = (V, I, O, f, g)$ находится в состоянии v_0. Тогда и только тогда для моделируемого этим автоматом программно-аппаратного СЗИ выполняется $m \in M_a$, когда существует последовательность переходов тестирования $\overline{s} \in S$ длины $len(\overline{s}) = l \in \mathbb{N}_0$ и для $\forall i \in I_{фб}$ выполняется одно из следующих условий:

1. i является автоматически вычислимой функцией безопасности в состоянии v_0;
2. i является автоматически вычислимой функцией безопасности в состоянии $v'_j \in V_{фб}$, $1 \leqslant j \leqslant l$ и $\overline{s}$ содержит последовательные переходы $s_0, \ldots, s_{j-1} \in T$ с началом в $v'_0 = v_0$, такие, что $i'_1, \ldots, i'_j$ являются автоматически вычислимыми стимулами в состояниях $v'_0, \ldots, v'_{j-1}$ соответственно.

Предположим, что $m \in M_a$, то есть программно-аппаратное СЗИ может быть проверено автоматически. Автомат, соответствующий данному СЗИ, $\widetilde{m} = (V, I, O, f, g)$ находится в начальном состоянии $v_0 \in V$. Так как программно-аппаратное СЗИ может быть проверено автоматически, то это означает что $\forall i \in I_{фб}$ можно проверить автоматически, то есть они являются автоматически вычислимыми функциями безопасности (в соответствии с Определением 5) в некоторых состояниях из $V_{фб}$ и эти состояния достижимы из v_0. То есть либо все i – автоматически вычислимые функции безопасности (в соответствии с Определением 5) в v_0, либо для тех i, которые не являются автоматически вычислимыми функциями безопасности в v_0, существуют автоматически вычислимые стимулы (в соответствии с Определением 6), которые позволяют перейти в соответствующие $v'_j \in V_{фб}$, в которых они являются автоматически вычислимыми функциями безопасности.

Если же $\forall i \in I_{фб}$ – автоматически вычислимые функции безопасности в v_0, то очевидно, что все множество функций безопасности можно проверить автоматически в состоянии v_0.

В противном случае очевидно, что из v_0 достижимы состояния $v'_j \in V_{фб}$, в которых автоматически вычислимы $\forall i \in I_{фб}$, то есть все множество функций безопасности можно проверить автоматически в таких состояниях v'_j.

Общий критерий возможности автоматизированного тестирования функций безопасности программно-аппаратных СЗИ:

Пусть автомат $\widetilde{m} = (V, I, O, f, g)$ находится в состоянии v_0. Тогда и только тогда для моделируемого этим автоматом программно-аппаратного СЗИ выполняется $m \in M_{па}$, когда выполняются следующие условия:

1. существует хотя бы одна $i \in I_{фб}$, удовлетворяющая условиям общего критерия возможности автоматического тестирования;
2. $\forall i' \in I_{фб} \setminus \{i\}$ удовлетворяет условиям возможности проведения ручного тестирования.

Данный случай полностью аналогичен предыдущим, так как в $M_{па}$ достаточно наличия всего одной автоматически вычислимой функции безопасности в некотором состоянии, которая достижима с помощью автоматически вычислимых стимулов.

Необходимо отметить, что по сути формализованные общие критерии применимости существующих способов тестирования к функциям безопасности программно-аппаратных СЗИ являются математическими определениями множеств M_p, M_a и $M_{па}$.

Сформулируем частные критерии для различных видов функций безопасности программно-аппаратных СЗИ. Если рассматривать функций безопасности программно-аппаратных СЗИ, разделяемых по виду взаимодействия с защищаемым СВТ в ИС, с точки зрения общих критериев возможности проведения ручного, автоматического и автоматизированного тестирования, то становится понятно, что множество переходов автомата для такой классификации СЗИ инициируется только стимулами, для которых требуется выполнение Условий 1-2 или 1-3.

Так как эти условия уже учитываются в определениях вычислимых (автоматически вычислимых) функций безопасности и стимулов, то для функций безопасности программно-аппаратных СЗИ, не взаимодействующих с ОС СВТ (выполняющихся независимо от ОС в составе СВТ и выполняющихся независимо от СВТ) и взаимодействующих со средой ОС СВТ, достаточно применения общих критериев.

Однако для функций безопасности, разделяемых по виду реализующей их аппаратной компоненты, стимулами, инициирующими переходы автомата, могут быть внешние по отношению к СЗИ функции, для которых требуется выполнение Условий 4 или 5 из раздела 2.2.1, которым не уделено достаточно внимания в общих критериях применимости способов тестирования. Поэтому необходимо детализировать предложенные общие критерии и учесть требования, которые характерны только для функций безопасности, реализованных на базе мобильной и стационарной компоненты соответственно, и тем самым получить частные критерии применимости.

Сначала сформулируем частный критерий возможности проведения ручного и автоматического тестирования для функций безопасности программно-аппаратных СЗИ, реализованных на базе мобильной аппаратной компоненты.

Частный критерий возможности проведения ручного и автоматического тестирования функций безопасности программно-аппаратных СЗИ, реализованных на базе мобильной аппаратной компоненты:

Пусть автомат $\widetilde{m} = (V, I, O, f, g)$ находится в состоянии v_0 и $m \in P_{моб}$. Тогда и только тогда $m \in M_p$ (или $m \in M_a$), когда при тестировании выполняются условия возможности проведения ручного тестирования (или условия общего критерия возможности автоматического тестирования), а также дополнительно следующее условие:

– $\forall v \in V_{фб}$: $\exists i_d \in I_{вн}$ – вычислимый стимул-функция ручного (или автоматического) отключения аппаратной компоненты от СВТ, а также $\exists v' \in V$ и в последовательности переходов тестирования $\overline{s}$ существует элемент $s = (v, i_d, o_d, v') \in T$. Существует хотя бы одно $v \in V_{фб}$: $\exists i_c \in I_{вн}$ – вычислимый стимул-функция ручного (или автоматического) подключения аппарат-

ной компоненты к СВТ, а также $\exists v'' \in V$ и в последовательности переходов тестирования $\overline{s}$ существует элемент $s' = (v'', i_c, o_c, v) \in T$.

Аналогично полученному частному критерию возможности проведения ручного и автоматического тестирования функций безопасности программно-аппаратных СЗИ, реализованных на базе мобильной аппаратной компоненты, можно сформулировать критерий для их автоматизированного тестирования. В данном случае к условиям 1–2 из общего критерия возможности автоматизированного тестирования добавится условие, полностью повторяющее условие из частного критерия возможности проведения ручного и автоматического тестирования функций безопасности программно-аппаратных СЗИ, реализованных на базе мобильной аппаратной компоненты.

Сформулируем частный критерий возможности проведения ручного и автоматического тестирования для функций безопасности программно-аппаратных СЗИ, реализованных на базе стационарной аппаратной компоненты.

Частный критерий возможности проведения ручного и автоматического тестирования функций безопасности программно-аппаратных СЗИ, реализованных на базе стационарной аппаратной компоненты:

Пусть автомат $\widetilde{m} = (V, I, O, f, g)$ находится в состоянии v_0 и $m \in P_{стац}$. Тогда и только тогда $m \in M_p$ (или $m \in M_a$), когда при тестировании выполняются условия возможности проведения ручного тестирования (или условия общего критерия возможности автоматического тестирования), а также дополнительно следующее условие:

– Существует хотя бы одно $v \in V_{фб}$: $\exists i_c, i_d \in I_{вн}$ – вычислимые стимулы-функции ручного (или автоматического) встраивания и отчуждения аппаратной компоненты в/из СВТ, а также $\exists v', v'' \in V$ и в последовательности переходов тестирования $\overline{s}$ существуют элементы $s = (v, i_d, o_d, v') \in T$ и $s' = (v'', i_c, o_c, v) \in T$.

Аналогично полученному частному критерию возможности проведения ручного и автоматического тестирования функций безопасности программно-аппаратных СЗИ, реализованных на базе стационарной аппаратной компоненты, можно сформулировать критерий для их автоматизированного тестирования. В данном случае к условиям 1–2 из общего критерия возможности автоматизированного тестирования добавится условие, полностью повторяющее условие из частного критерия возможности проведения ручного и автоматического тестирования функций безопасности программно-аппаратных СЗИ, реализованных на базе стационарной аппаратной компоненты.

В соответствии с этим для функций безопасности программно-аппаратных СЗИ, разделяемых по виду реализующей их аппаратной компоненты, необходимо вместо условий возможности ручного и автоматического тестирования применять условия из частных критериев возможности ручного и автоматического тестирования в зависимости от типа аппаратной компоненты (мобильной или стационарной соответственно). Суть же данных частных критериев состоит в выполнении Условий 4 или 5 в части осуществления в ходе тестирования подключения и отключения аппаратной компоненты к/от СВТ. Кроме того данные критерии уточняют состав

множества всех состояний V для программно-аппаратных СЗИ с функциями безопасности, реализованными на базе мобильной или стационарной аппаратной компоненты.

2.3. Алгоритм тестирования функций безопасности программно-аппаратных СЗИ, основанный на использовании разработанной модели программно-аппаратных СЗИ

Ранее в разделе 2.2 сформированы критерии о принципиальной возможности проведения тестирования программно-аппратных СЗИ, с помощью которых становится возможным выполнить условие (2.1) для решения задачи тестирования из Определения 2. Однако, выполнение предложенных ранее критериев не обеспечивает полноты тестирования, так как проход по состояниям программно-аппаратного СЗИ может не обходить абсолютно все состояния из $V_{фб}$, а также не учитывать всевозможные стимулы для обхода таких состояний. То есть выполнение только условия (2.1) для решения задачи тестирования программно-аппаратных СЗИ является необходимым, но не достаточным. В данном разделе предлагается подход к выполнению оставшихся условий (2.2)–(2.3) для решения задачи тестирования программно-аппаратных СЗИ, обеспечивающий полноту и оптимальность тестирования, и тем самым показывающий каким именно образом необходимо проводить тестирование. Для этого можно воспользоваться существующими положениями из теории графов [2,81,82], представив некоторое программно-аппаратное СЗИ $m \in M$ в виде графа [10,51].

Обозначим $G_m = (V, E)$ – ориентированный граф без петель и кратных дуг (простой орграф), соответствующий программно-аппаратному СЗИ $m \in M$, представленному в виде конечного детерминированного автомата $\widetilde{m} = (V, I, O, f, g)$ [106], где:

– V – множество вершин графа, соответствующих состояниям автомата;

– $E \subseteq V \times I \times O \times V$ – множество ориентированных ребер (дуг) графа, помеченных стимулами и реакциями, то есть переходов $s = (v, i, o, v') \in T$ автомата.

Далее будем предполагать, что для определенного выше графа $G_m = (V, E)$ выполняются один из критериев принципиальной возможности провести тестирование из раздела 2.2, а множество дуг E содержит только переходы, в которых участвуют вычислимые стимулы и функции безопасности в соответствующих состояниях из V. Это также означает, что каждая дуга (v, i, o, v') не образует петлю, то есть $v \neq v'$, так как в противном случае i не является вычислимой функцией безопасности или стимулом.

Из условия (2.2) Определения 2 следует, что обход графа $G_m = (V, E)$ должен осуществляться преимущественно по вершинам из множества $V_{фб} \subseteq V$, а не вообще по всему множеству вершин V. Поэтому вместо графа $G_m = (V, E)$ будем рассматривать производный от него граф G'_m, который будет построен с помощью удаления из оригинального графа неиспользуемых при решении задачи тестирования некоторых вершин и дуг.

Обозначим $G'_m = (V', E')$ – граф, полученный из G_m в результате удаления вершин $v \notin V_{фб} \cup \{v_0\}$ и некоторых дуг из E в соответствии со следующими правилами [106]:

– Если для $v \in V \backslash (V_{фб} \cup \{v_0\})$ существуют $v', v'' \in V \backslash \{v\}$: $v' \neq v''$, $(v, v'') \in E$ и $(v', v) \in E$, но $(v', v'') \notin E$ – необходимо для всех таких v', v'' добавить новые дуги (v', v'') в множество E (пометив их объединенными стимулами и реакциями дуг (v, v'') и (v', v)), удалить дуги (v, v'') и

(v', v) из E, удалить v из V (то есть если через неиспользуемую вершину проходит путь между двумя другими вершинами, которого еще нет в графе, нужно вместо входящей и исходящей дуги каждого такого пути добавить одну дугу с тем же началом и концом, после чего удалить саму вершину).

– Если для $v \in V \setminus (V_{фб} \cup \{v_0\})$ из любой другой $v' \in V \setminus \{v\}$ существует входящая дуга $(v', v) \in E$ и не существует $v'' \in V \setminus \{v\}$ с исходящей дугой $(v, v'') \in E$ – необходимо удалить все дуги (v', v) из E, удалить v из V (то есть если в неиспользуемую вершину есть только входящие дуги, но нет исходящих, нужно удалить эти дуги, после чего удалить саму вершину).

– Если для $v \in V \setminus (V_{фб} \cup \{v_0\})$ не существует $v' \in V \setminus \{v\}$ с дугами $(v, v') \in E$ или $(v', v) \in E$ – необходимо удалить v из V (то есть если неиспользуемая вершина не имеет входящих в нее или исходящих из нее дуг, нужно удалить эту изолированную вершину);

– Если для $v \in V \setminus (V_{фб} \cup \{v_0\})$ из любой другой $v' \in V \setminus \{v\}$ не существует входящей дуги $(v', v) \in E$ и существует $v'' \in V \setminus V_{фб}$ с исходящей дугой $(v, v'') \in E$, для которой не существует $v''' \in V \setminus \{v, v''\}$ с дугой $(v'', v''') \in E$ – необходимо удалить все дуги (v, v'') из E, и если не существует $v^* \in V_{фб}$ с дугой $(v, v^*) \in E$ – удалить v из V (то есть если из неиспользуемой вершины есть только исходящие дуги, то дуги, входящие в другую неиспользуемую вершину и не являющиеся промежуточными дугами для некоторого пути, нужно удалить, и если из неиспользуемой вершины больше не осталось дуг, нужно удалить саму вершину).

Необходимо отметить, что при удалении дуг из графа G_m при выполнении каждого из правил, кроме первого, соответствующие данным дугам стимулы и реакции исключаются из дальнейшего рассмотрения при решении задачи тестирования. В первом правиле стимулы и реакции удаляемых дуг объединяются и используются в дальнейшем для формирования новых дуг, участвующих в решении задачи тестирования.

Пример графа G_m и результат удаления неиспользуемых вершин и дуг по указанным выше правилам для получения графа G'_m приведены на Рисунках 2.2–2.3.

Утверждение 1. *Построенный граф $G'_m = (V', E')$ эквивалентен графу $G_m = (V, E)$ с точки зрения решения задачи тестирования из Определения 2 в части условий* (2.1)–(2.3).

Доказательство. В соответствии с построением G'_m – из исходного графа G_m не удаляются следующие вершины и дуги:

– v_0;

– вершины, для которых есть потенциально вычислимые функции безопасности;

– дуги в вершины с потенциально вычислимыми функциями безопасности (в первом правиле такие дуги могут заменяться на аналогичные без участия промежуточной удаляемой вершины).

При указанном удалении вершин и дуг в графе G'_m не нарушается связность [2, 81, 82] для сооветствующих вершин G_m. Вершина v_0 не удаляется, а все аналогичные пути из v_0 в вершины из $V_{фб}$ будут присутствовать и в E', то есть в G'_m сохраняется выполнение критериев из раздела 2.2 и условия (2.1) из Определения 2.

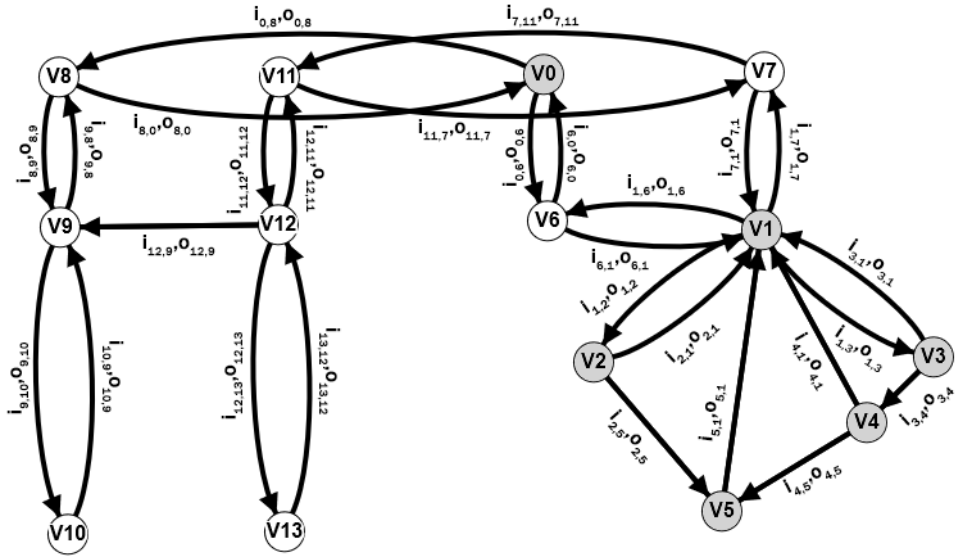


Рисунок 2.2 – Граф G_m, где выделенные вершины $v_0, \ldots v_5$ – начальное состояние и состояния с потенциально вычислимыми функциями безопасности, остальные вершины не рассматриваются для решения задачи тестирования

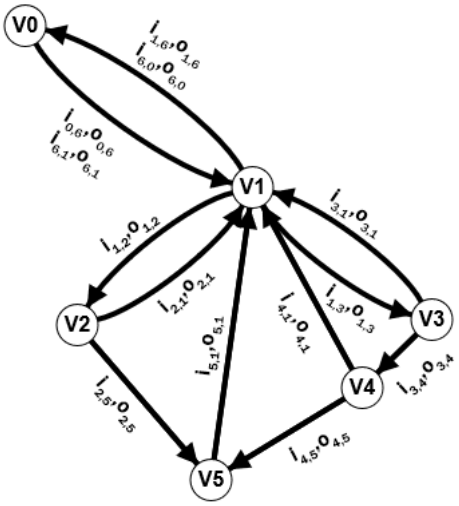


Рисунок 2.3 – Граф G'_m, полученный из G_m с помощью удаления из оригинального графа неиспользуемых при решении задачи тестирования некоторых вершин и дуг

Если для графа G_m выполняется условие (2.2) из Определения 2, то в графе G'_m они также будут выполняться (и наоборот), так как из G_m не удаляются дуги в вершины с потенциально вычислимыми функциями безопасности.

Выполнение условия (2.3) из Определения 2 не нарушается, так как если множество совершенных при тестировании переходов минимально для G_m, то оно также минимально и для G'_m и наоборот. □

В соответствии с Утверждением 1 решение задачи тестирования в части выполнения условий (2.1)–(2.3) для графа G'_m будет являться решением задачи тестирования для изначального графа G_m. Рассмотрим задачу тестирования с точки зрения известных положений теории графов [101, 112] – условия (2.2)–(2.3) определения возможности решения задачи тестирования из Определения 2 заключаются в поиске пути, проходящему через все дуги графа G'_m хотя бы по одному разу за минимальное количество переходов. Таким образом, в соответствии с [81, 101, 112] задача тестирования сводится к задаче китайского почтальона (англ. Chinese postman problem), также известной как задача инспекции дорог (англ. Route Inspection Problem), либо, как частный случай – к задаче поиска Эйлерова пути (англ. Eulerian path / Eulerian trail).

Эйлеров путь – оптимальное решение данной задачи, так как каждая дуга обходится ровно один раз. Однако Эйлеров путь в графе G'_m существует тогда и только тогда, когда [81, 106, 112]:

1. Любая вершина $v \in V$ графа G'_m либо принадлежит орцепи, либо лежит в компоненте сильной связности [2, 81, 82, 101]. Общий вид графа G'_m для решения задачи тестирования представлен на Рисунке 2.4.

2. Только для двух вершин $v', v'' \in V'$ полустепени захода и выхода [112] удовлетворяют следующим условиям:

$$outdeg(v') - indeg(v') = 1;$$

$$outdeg(v'') - indeg(v'') = -1.$$

3. Для остальных вершин выполняется: $\forall v \in V' \setminus \{v', v''\}$: $indeg(v) = outdeg(v)$.

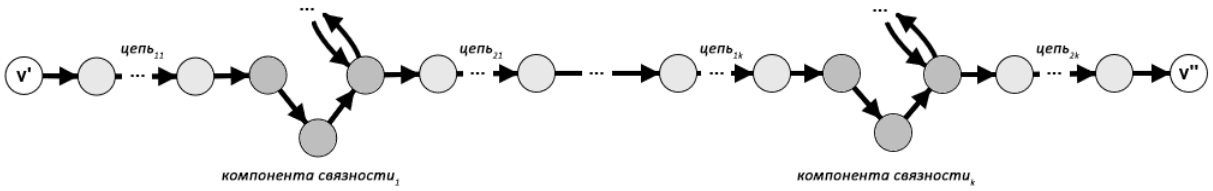


Рисунок 2.4 – Общий вид графа для решения задачи тестирования – отдельные вершины, цепи или компоненты связности могут отсутствовать, $k \in \mathbb{N}_0$

В случае, если в графе не существует Эйлерова пути, то условия (2.2)–(2.3) могут быть выполнены с помощью решения задачи китайского почтальона или инспекции дорог, минимизирующей число дуг графа, которые необходимо пройти повторно. Эту задачу также можно решить только если любая вершина графа G'_m либо принадлежит орцепи, либо лежит в компоненте сильной связности. Дополнительно к этому в графе не должно быть циклов с отрицательным общим

весом (в графе G_m и G'_m вес каждой дуги равен 1 и существование таких циклов невозможно). Если данные условия не выполняются, то задача тестирования не может быть решена, так как невозможно выполнить условие (2.2).

Таким образом, можно предложить следующий алгоритм решения задачи тестирования функций безопасность программно-аппаратного СЗИ [108]:

1. Построить из изначального графа программно-аппаратного СЗИ G_m соответствующий граф G'_m с помощью удаления неиспользуемых при решении задачи тестирования некоторых вершин и дуг, используя приведенные ранее правила.

2. Если в G'_m есть изолированные вершины (не удаленные при построении графа G'_m) – задача тестирования не может быть решена, так как невозможно будет выполнить условие (2.2).

3. Необходимо проверить, что в G'_m любая вершина $v \in V$ либо принадлежит орцепи, либо лежит в компоненте сильной связности. Например, с использованием алгоритма Косараджу–Шарира за два обхода графа в глубину [2, 81, 82, 111], представленного на Рисунках 2.5 и 2.6, можно найти все компоненты связности и проверить связанность начальной вершины v_0 со всеми остальными вершинами. В противном случае задача тестирования не может быть решена, так как невозможно будет выполнить условие (2.2).

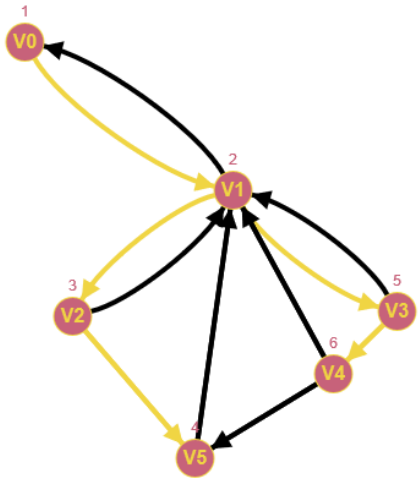


Рисунок 2.5 – Результат обхода графа G'_m в глубину из начальной вершины v_0, все вершины достижимы из v_0 – граф G'_m, по крайней мере, слабо связный

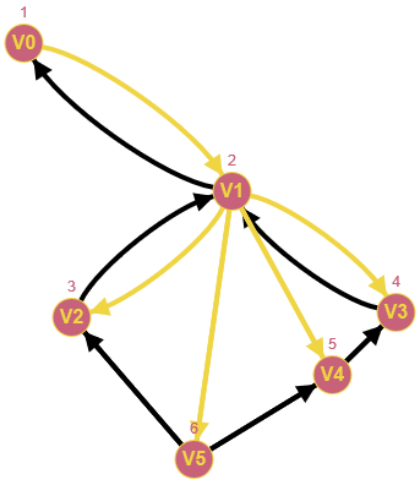


Рисунок 2.6 – Результат обхода обратного графа G'_m в глубину из начальной вершины v_0, все вершины достижимы из v_0 – граф G'_m сильно связный (состоит из одной компоненты связности)

4. Для всех вершин графа G'_m необходимо вычислить разницу полустепеней выхода от полустепеней входа, например, как на Рисунке 2.7. Если в графе есть только сбалансированные вершины (разница равна 0), то в графе существует Эйлеров цикл. Если в графе есть только две несбалансированные вершины с разницей 1 и -1, а остальные вершины сбалансированные, то в графе существует Эйлеров путь. В этих случаях необходимо продолжить алгоритм с пункта 9.

5. В противном случае необходимо рассмотреть отдельно несбалансированные вершины с целью нахождения путей, которые необходимо пройти повторно с сохранением требования по

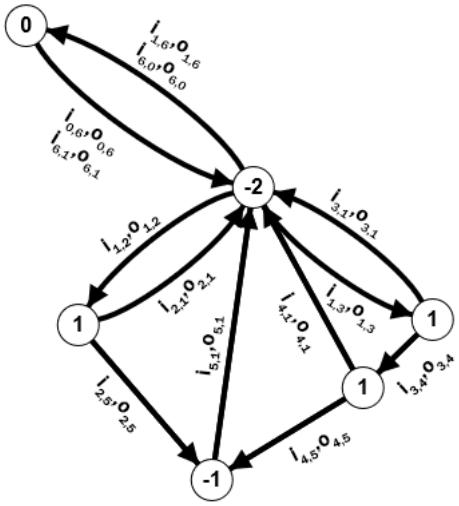


Рисунок 2.7 – Граф G'_m со значениями разницы полустепеней выхода и захода для каждой вершины. G'_m не Эйлеров граф, так как в v_1 входящих дуг на 2 больше чем исходящих

минимизации обхода дуг из условия (2.3). Несбалансированные вершины можно представить в виде биграфа, изображенного на Рисунке 2.8.

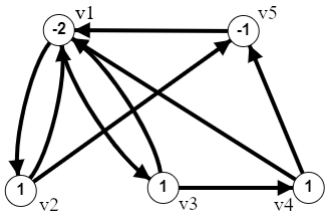


Рисунок 2.8 – Биграф, являющийся подграфом графа G'_m, внизу вершины с положительной разницей полустепеней выхода и захода, вверху – с отрицательной

6. С помощью алгоритма поиска кратчайших путей для всех вершин в полученном биграфе (например, с помощью алгоритма Флойда–Уоршелла [2,81]) вычислить длину кратчайших путей от вершин с отрицательной разницей полустепеней выхода и входа к вершинам с положительной разницей, пример приведен на Рисунке 2.9.

7. Выбрать с использованием, например, Венгерского алгоритма (алгоритма Куна-Манкреса) или алгоритма Форда-Фалкерсона [2, 81, 82], из всех сочетаний возможных кратчайших путей те пути, при повторном использовании которых все вершины станут сбалансированными, но при этом суммарная длина этих путей будет минимальна. Пример приведен на Рисунке 2.10.

При добавлении пути от вершины с отрицательной разницей полустепеней выхода и захода к вершине с положительной разницей, разница в обеих вершинах изменится ровно на 1 (для первой увеличится на 1, для второй уменьшится на 1). При этом для всех остальных вершин разница не изменится, так как могут добавиться только входящая и исходящая дуга (суммарная разница останется 0).

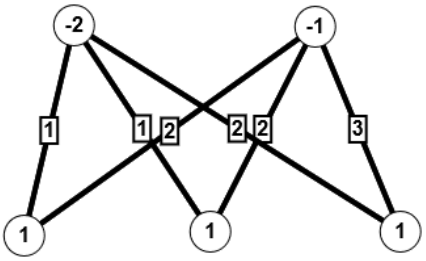


Рисунок 2.9 – Биграф, ребра которого подписаны длиной кратчайших путей от вершин с отрицательной разницей полустепеней выхода и входа до вершин с положительной разницей

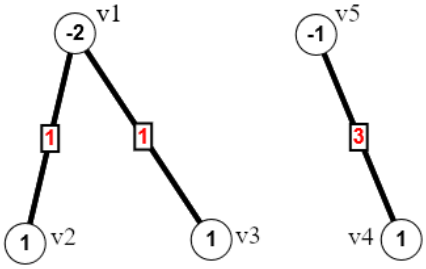


Рисунок 2.10 – Один из оптимальных вариантов путей с повторным проходом по некоторым дугам, в случае добавления таких путей в G'_m все вершины станут сбалансированными.

8. Добавить дуги для выбранных на предыдущем шаге дополнительных путей в граф G'_m. Так как теперь все вершины сбалансированы, то будет существовать Эйлеров цикл, как в графе на Рисунке 2.11.

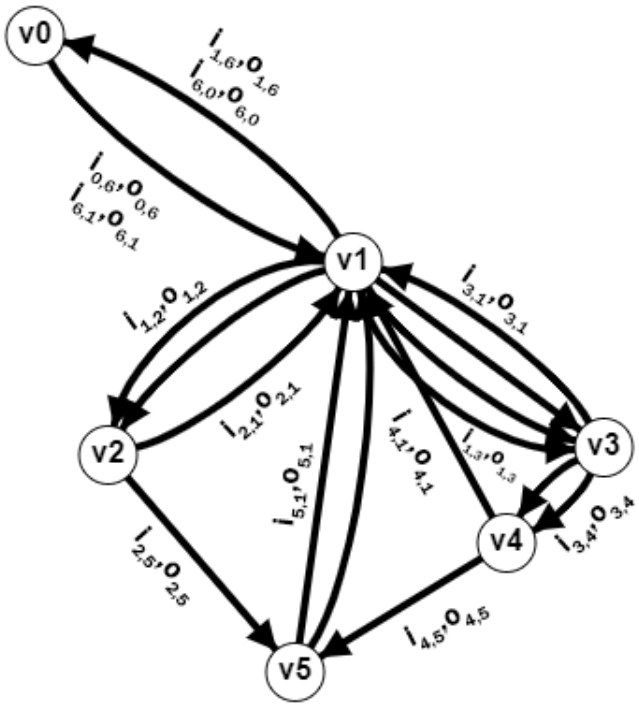


Рисунок 2.11 – Граф G'_m с дополнительными дугами – Эйлеров граф, так как все вершины сбалансированы

9. С помощью алгоритма на основе циклов, также известного как алгоритм Хиерхольцера [112], построить Эйлеров цикл в полученном графе. Посещение достроенных на предыдущем шаге дуг эквивалентно повторному посещению соответствующих дуг графа G'_m. Некоторые итерации Эйлерова цикла можно менять местами, например промежуточные строки на Рисунке 2.12 можно ставить в произвольном порядке.

Блок-схема предложенного алгоритма тестирования приведена на Рисунке 2.13.

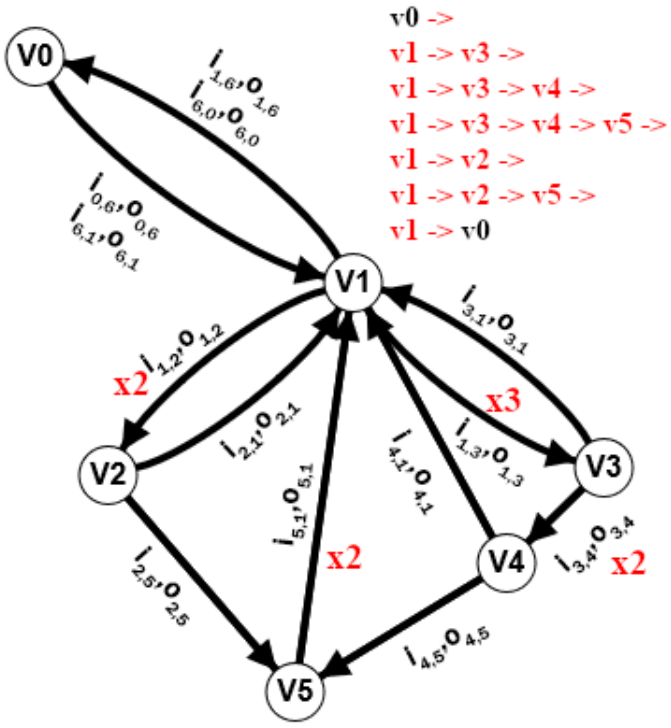


Рисунок 2.12 – Один из оптимальных вариантов обхода всех дуг графа G'_m, для помеченных дуг обход осуществляется более одного раза

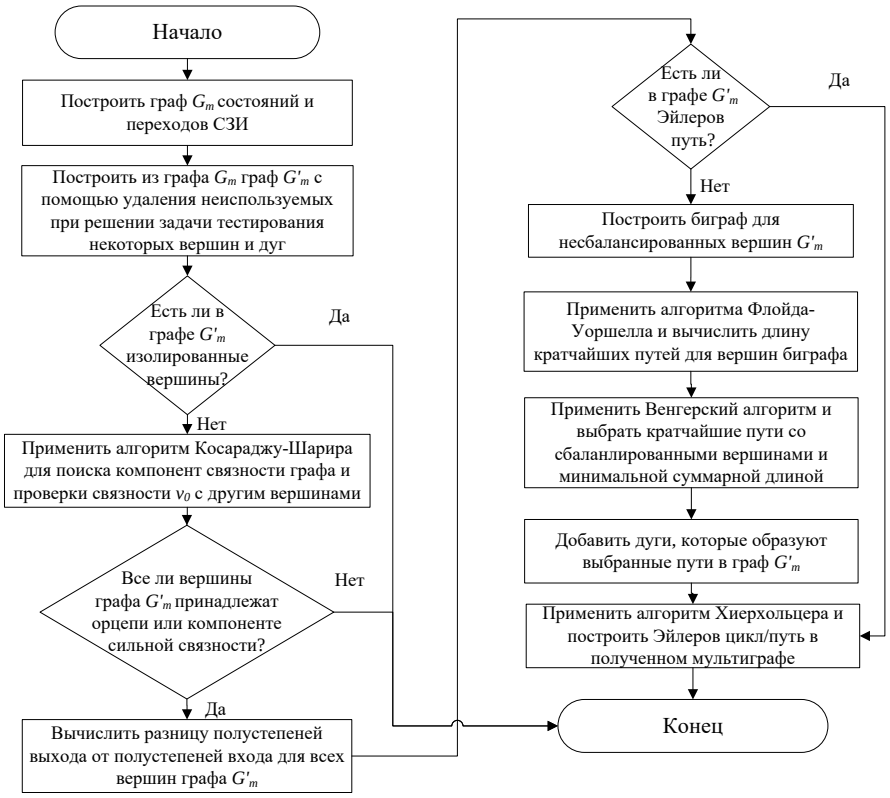


Рисунок 2.13 – Блок-схема алгоритма решения задачи тестирования функций безопасности программно-аппаратных СЗИ

Следствие 1. Решение задачи тестирования из Определения 2 существует тогда и только тогда, когда любая вершина $v \in V$ соответствующего графа G'_m либо принадлежит орцепи, либо лежит в компоненте сильной связности. При этом алгоритм решения задачи тестирования имеет полиномиальную сложность $O(|V|^3)$.

Доказательство. Ранее было показано, что алгоритм решения задачи тестирования заканчивается успешно тогда и только тогда, когда любая вершина $v \in V$ графа G'_m либо принадлежит орцепи, либо лежит в компоненте сильной связности (шаги 1–3 алгоритма). В противном случае задача тестирования не имеет решения.

Сложность представленного алгоритма будет равна максимальной сложности используемых в нем известных алгоритмов на графах, так как все шаги алгоритма выполняются последовательно и для них не используется вложенность. Сложности используемых алгоритмов следующие [106]:

– сложность построения графа G'_m для представления графа в виде матрицы смежности $O(|V'|^3)$. За $O(|V'|^3)$ выполняется первая проверка при построении G'_m, за $O(|V'|^2)$ – вторая и третья, а четвертая проверка – за $O(|V'|^3)$. Соответственно общая сложность не будет превосходить сложности первой или четвертой проверки.

– Алгоритм Косараджу–Шарира – $O(|V'|^2)$ для матрицы смежности [2, 81, 82, 111];

– рассчет полустепеней выхода и входа и их разницы для каждой вершины – $O(|V'|^2)$ для матрицы смежности [2, 81, 82];

– Алгоритм Флойда–Уоршелла – $O(|V'|^3)$ [2, 81];

– Венгерский алгоритм – $O(|V'|^3)$ [2, 81, 82];

– Алгоритм Хиерхольцера – $O(|E'|)$ [112].

То есть в случае если G'_m содержит Эйлеров путь (цикл) или не содержит – сложность алгоритма будет $O(|V'|^3)$ для представления графа в виде матрицы смежности. То есть задача тестирования программно-аппаратных СЗИ принадлежит классу задач P, а не NP [2, 82] и существует алгоритм, решающий эту задачу за полиномиальное время [101, 112]. □

Предложенный алгоритм подразумевает обход вершин графа с помощью одной последовательности переходов. В случае, если обход графа по приведенному алгоритму невозможно выполнить с помощью одной последовательности переходов, то это означает, что в графе есть несколько несвязанных ветвей. А это значит, что в СЗИ есть ошибка, так как нельзя добиться, чтобы все функции безопасности выполнялись корректно без нарушения работы других функций безопасности. Поэтому в работе не рассматривается вариант с несколькими последовательными обходами графа СЗИ.

Таким образом, в разделе предложен подход к проверке выполнимости задачи тестирования произвольного программно-аппаратного СЗИ с использованием положений теории графов. В соответствии с данным подходом для некоторых программно-аппаратных средств защиты эту задачу решить невозможно (не все вершины графа принадлежат орцепи или компоненте сильной связности), для остальных СЗИ задача имеет решение в худшем случае за полиномиальное время с использованием известных алгоритмов на графах.

Необходимо также отметить, что задача тестирования может быть изменена и включать в себя негативное тестирование, то есть проверку «невыполнимости» функций безопасности в состояниях $V \setminus V_{фб}$. Подходы и алгоритмы решения такой задачи тестирования являются полностью аналогичными, за исключением необходимости их применения ко всему графу G_m, а не к производному графу G'_m, полученному после исключения ненужных для изначальной задачи тестирования некоторых вершин и дуг.

2.4. Алгоритм верификации функий безопасности программно-аппаратных СЗИ, основанный на использовании разработанной модели программно-аппаратных СЗИ

Решив задачу тестирования программно-аппаратного средства защиты информации, например, с использованием алгоритма из раздела 2.3, необходимо выполнить верификацию полученных результатов – перечня зафиксированных ошибок и особенностей (см. раздел 1.4). Для этого необходимо выполнить классификацию зафиксированных ошибок, которая включает определение типа ошибки, возможности ее компенсации, а также степень ее критичности [36, 39]. В результате всех циклов тестирования на разных наборах входных данных необходимо составить итоговую таблицу верификации программно-аппаратного СЗИ, которая используется для анализа работоспособности продукта в целом и принятия решения об успешном завершении тестирования и верификации.

Некоторые ошибки, найденные в процессе тестирования, могут быть исправлены достаточно быстро, еще до окончания верификации. Учитывая, что каждое внесенное исправление в один из модулей программной или аппаратной компоненты СЗИ, может повлечь за собой изменение работы других модулей, необходимо провести повторное тестирование исправленного продукта по всей ПМИ. Таким образом, постоянное изменение тестируемого и верифицируемого объекта может привести к путанице в результатах и к так называемому «бесконечному тестированию», что, в свою очередь, приведет к задержке выпуска СЗИ. Для предотвращения входа в бесконечный цикл тестирования необходимо либо принять решение о внесении исправлений в следующую версию и завершить верификацию текущей, с выносом вердикта относительно ее выпуска, либо остановить тестирование и верификацию текущей версии и сразу приступить к тестированию новой, максимально доработанной на текущий момент версии программно-аппаратного СЗИ и его функций безопасности.

Для СЗИ, в том числе программно-аппаратных, процесс верификации имеет некоторые особенности, касающиеся анализа влияния ошибок, обнаруженных при выполнении реализованных функций безопасности, на защищенность системы, в которой эти средства используются (см. раздел 1.4 и [36]). Такие особенности обязательно должны быть учтены в программе и методике испытаний, без них тестирование функций безопасности СЗИ нельзя считать полным.

Ошибки, выявленные при тестировании функций безопасности СЗИ, необходимо отнести к одному из следующих видов: несущественные опечатки и ошибки (например, опечатка в выводимом сообщении или некорректное название какой-либо функции), не влияющие на корректность выполнения функций безопасности; ошибки, приводящие к неработоспособности одной

или нескольких функций безопасности СЗИ; ошибки, приводящие к неработоспособности или нарушению защищенности системы, в которой используется СЗИ.

Перечисленные виды ошибок неравнозначны с точки зрения работы СЗИ, поэтому необходима определенная градация всех найденных ошибок относительно их влияния на выполнение основной задачи – защиты информационных ресурсов и обеспечения безопасности.

Шкала критичности ошибок индивидуальна и, как правило, предназначена для внутреннего использования компанией, производящей конкретное средство защиты. При этом корректность оценки критичности ошибок является основополагающим фактором при принятии решения о возможности финализации и выпуска СЗИ.

Рассмотрим пример шкалы критичности ошибок, найденных при тестировании функций безопасности программно-аппаратного СЗИ. Ошибки можно разделить на несколько типов [36]:

1. Ошибки интерфейса. К ним относятся недочеты в удобстве пользовательского интерфейса, корректности отображения всех его элементов, опечатки в системных сообщениях и тому подобное. Эти ошибки не влияют на выполнение функций безопасности СЗИ, функциональность и защищенность системы, в которой применяется СЗИ. Их исправление необходимо для обеспечения комфортной работы конечного пользователя. Наличие таких недочетов не опасно для защищаемой системы, соответственно им присваивается минимальный уровень критичности.

2. Ошибки, ограничивающие функциональность СЗИ, без нарушения его функций безопасности. Этот тип ошибок накладывает некоторые ограничения на функциональность СЗИ, при этом не подвергая опасности защищаемую систему. В этом случае либо некорректно функционирующие возможности СЗИ не отвечают непосредственно за безопасность защищаемой системы, либо отсутствие этих функций можно компенсировать путем применения других средств и мер без снижения уровня защищенности.

3. Ошибки, связанные с нарушением функций безопасности СЗИ. К этому типу относятся ошибки, способные повлиять на защищенность системы или данных из-за нарушения функций безопасности СЗИ, которое создает предпосылки для успешной реализации атаки с использованием возникшей уязвимости. Они являются наиболее критичными и наличие даже одной ошибки этого типа может привести к завершению верификации запретом финализации и выпуска продукта, кроме тех случаев, когда это нарушение можно компенсировать дополнительными средствами и мерами, например, донастройкой ОС.

Необходимо отметить, что тестировщик не всегда может определить тип ошибки на основании обнаруженных ее проявлений без участия разработчика. Поэтому привлечение разработчика к анализу обнаруженных проявлений ошибок является обязательным условием, без которого нельзя точно зафиксировать ошибки, и, если этого не сделать, то это может повлечь за собой неверную их классификацию. Например, если ошибка, найденная тестировщиком, проявляется в том, что при проверке ЭП файла, в который внесены изменения после ее простановки, выдается сообщении о корректности этой ЭП, то это может быть ошибка второго типа, когда неверно отрабатывает функция формирования сообщения о результатах проверки подписи. Однако к такому проявлению может приводить и ситуация, когда, некорректно работает функция проверки

ЭП. В данном случае это уже будет ошибка третьего типа, которая как раз может привести к нарушению безопасности системы и возможности реализации каких-либо атак со стороны потенциального злоумышленника. В описанном случае тестировщик самостоятельно не сможет разобраться, в чем именно суть возникшего проявления ошибки, и необходимо привлечение разработчика для ее анализа и фиксации.

После классификации ошибок и особенностей, а также анализа полученных результатов необходимо подвести итог – выполняются ли все требования, предъявленные к функциям безопасности данного СЗИ, или есть критические ошибки, приводящие к их нарушению (даже с учетом компенсационных мер), и не позволяющие принять решение о начале финализации. Если таких ошибок нет, то принимается решение о выпуске СЗИ, иначе – о задержке для последующей доработки, а также повторного тестирования и верификации.

С одной стороны, для принятия решения о возможности выпуска СЗИ можно руководствоваться наличием/отсутствием критических ошибок СЗИ, как это описано выше. С другой стороны, можно использовать более сложные методы [96], например, из теории оптимизации и принятия решений, в которых данная задача может рассматриваться как задача принятия решения в условии полной определенности, то есть когда наличие определенного типа ошибки ведет к «ухудшению» СЗИ в некоторой известной степени. Для решения такой задачи можно использовать, например, известный алгоритм простого выбора, основанного на методах Саати, Коггера и Ю (англ. Saaty, Cogger and Yu method) [94], оценивая СЗИ (именуемые альтернативами) по критериям оценки с учетом «важности» каждого из них, рассчитывая при этом некоторую сумму – значение обобщенного критерия. Решать поставленную задачу с помощью такого математического аппарата возможно вследствие того, что [33, 105]:

– описанная задача является задачей многокритериального выбора в условиях определенности с малым числом критериев и альтернатив;

– в качестве критериев оценки можно использовать как качественные (наличие/отсутствие ошибки), так и количественные характеристики (количество тех или иных ошибок), то есть при необходимости цель задачи может быть изменена без изменения применяемого аппарата;

– при фиксации ошибок во время тестирования всегда можно выявить необходимую дополнительную информацию (количество ошибок, степень их «важности» и так далее);

– для тестирования и верификации важно получить однозначный ответ – возможно или нет осуществить выпуск СЗИ (например, зная некоторую допустимую грань критичности ошибок в СЗИ), а данные методы больше всего подходят в данном случае, в то время как другие могут предложить множество оптимальных ответов (так называемое, множество Парето [94]).

Применим алгоритм простого выбора, основанный на методах Саати, Коггера и Ю, для задачи принятия решения о возможности выпуска СЗИ. Данный алгоритм предполагает первоначально построить вектор весов (весовых коэффициентов) для критериев оценки. В нашем случае критерии оценки f_1, f_2 и f_3 – наличие ошибок трех уровней критичности. Вектор весов для данных критериев оценки $\alpha = (\alpha_1, \alpha_2, \alpha_3)$, обладающий свойством нормированности ($\sum_{i=1}^{3} \alpha_i = 1$) [105].

Для этого по заданной шкале критичности вначале определяются отношения попарного превосходства критериев оценки между собой – $\alpha_{12} = \alpha_1/\alpha_2, \alpha_{13} = \alpha_1/\alpha_3, \alpha_{23} = \alpha_2/\alpha_3$. В данном случае делается переход от качественных характеристик критериев к количественным отношениям, а затем с помощью решения линейного уравнения, полученного из условия нормированности, определяются сами значения весовых коэффициентов α_1, α_2 и α_3.

Далее аналогично рассчитываются значения частных критериев, соответствующих альтернативам – $\alpha^{(1)} = (f_1(alt_1), \ldots, f_1(alt_k))$, $\alpha^{(2)} = (f_2(alt_1), \ldots, f_2(alt_k))$ и $\alpha^{(3)} = (f_3(alt_1), \ldots, f_3(alt_k))$, где $alt_1, \ldots, alt_k, k \in \mathbb{N}$ – оцениваемые альтернативы (СЗИ). На основе качественных и количественных характеристик альтернатив делается переход к количественным отношениям – $\alpha^1_{12} = \alpha^1_1/\alpha^1_2, \ldots \alpha^1_{k-1k} = \alpha^1_{k-1}/\alpha^1_k, \alpha^2_{12} = \alpha^2_1/\alpha^2_2, \ldots, \alpha^2_{k-1k} = \alpha^2_{k-1}/\alpha^2_k, \alpha^3_{12} = \alpha^3_1/\alpha^3_2, \ldots, \alpha^3_{k-1k} = \alpha^3_{k-1}/\alpha^3_k$. После чего, учитывая условие нормированности векторов $\alpha^{(1)}$, $\alpha^{(2)}$ и $\alpha^{(3)}$, решается система линейных уравнений и определяются значения $f_1(alt_1)$, $f_2(alt_1)$, $f_3(alt_1)$, $\ldots f_1(alt_k)$, $f_2(alt_k)$, $f_3(alt_k)$ – весовые коэффициенты значимости того или иного вида ошибок в конкретной альтернативе [33, 105].

Для оценки критичности ошибок в альтернативах необходимо вычислить и сравнить значения обобщенных критериев по следующей формуле линейной свертки: $J(alt_i) = \sum\limits_{j=1}^{3} \alpha_j f_j(alt_i), i = 1, \ldots, k, k \in \mathbb{N}$. Большее значение обобщенного критерия соответствует наличию больших по общей критичности ошибок в той или иной альтернативе, поэтому задача сводится к минимизации значения обобщенного критерия.

С использованием описанного математического аппарата возможно сравнивать между собой различные версии одного и того же СЗИ, либо использовать определенную версию как эталон и выпускать средство защиты только при показателях, близких или превышающих показатели эталона. Однако данная оценка всегда может быть неточной из-за присутствия невыявленных в ходе тестирования СЗИ ошибок. Кроме того, можно ввести новый уровень иерархии критериев оценки – критичность самих функций безопасности, в которых выявлены ошибки того или иного уровня критичности, для решения такой задачи можно использовать прежний математический аппарат и аналогичные расчеты.

На основании вышесказанного можно предложить алгоритм верификации программно-аппаратных СЗИ, реализующих функции безопасности, основанной на классификации обнаруженных ошибок, анализе степени их критичности и влияния на защищенность системы или данных. Этот алгоритм имеет широкое назначение и применим не только для всех выделенных в разделе 1.2 видов программно-аппаратных СЗИ, но также и для программных средств защиты. Однако данный алгоритм неактуален для ПО или любых других программно-аппаратных средств, так как он рассматривает критичность ошибок именно функций безопасности, которые отсутствуют в любых других средствах, не предназначенных для защиты информации. Данный алгоритм позволяет оценить критичность ошибок, возникающих при выполнении вычислимых по критериям из раздела 2.2 функций безопасности (то есть в тех случаях, когда эти функции безопасности вычислимы, но выход автомата соответствует ошибке), выполнить анализ полученных результатов и степени влияния ошибок на защищенность системы, на основании чего принять решение об успешном завершении тестирования или о возврате СЗИ на доработ-

ку. Согласно предложенному алгоритму верификации программно-аппаратных СЗИ необходимо [33, 105]:

1. Сформировать шкалу критичности ошибок для верифицируемого программно-аппаратного СЗИ (например, в соответствии с предложенным выше разделением ошибок на типы: несущественные и не влияющие на корректность выполнения функций безопасности; приводящие к неработоспособности одной или нескольких функций безопасности; приводящие к неработоспособности или нарушению защищенности системы, в которой используется СЗИ).

2. Выявить и зафиксировать вычислимые функции безопасности, результат выполнения которых отрицателен.

3. Выявить и зафиксировать из оставшихся функций безопасности те, которые не являются вычислимыми в результате отрицательного завершения проверок из пункта 2.

4. Провести классификацию ошибок, полученных в результате отрицательного завершения функций безопасности из пунктов 2 и 3 в соответствии с выбранной шкалой критичности ошибок из пункта 1.

5. Принять решение о возможности успешного завершения тестирования или необходимости исправления выявленных ошибок с использованием положений теории оптимизации и принятия решений (например, алгоритма простого выбора, основанного на методах Саати, Коггера и Ю) [94], либо на основе других оценок.

6. Тестирование, а также пункты 1-5 проводить для зафиксированной версии программно-аппаратного СЗИ (программной и аппаратной компонент), в случае внесения любых изменений в средстве защиты или исправления выявленных ошибок до окончания процесса верификации – начать заново с тестирования и затем перейти к пункту 2.

Блок-схема предложенного алгоритма верификации приведена на Рисунке 2.14 [105].

Преимущество данного алгоритма состоит в том, что в отличие от принятого в известных работах и нормативных документах вероятностного подхода к управлению рисками и оценке надежности ИС (см. раздел 1.4, а также [24, 95]) данный алгоритм предлагает детерминированный подход, наиболее подходящий для целей верификации программных и программно-аппаратных СЗИ. Верификация СЗИ должна проводиться непосредственно до их внедрения в ИС, при этом должна определяться не вероятность отказа ИС или его частота, а целесообразность исправления уже выявленных конкретных ошибок, которые могут проявится в ходе работы системы и повлиять на ее защищенность, непосредственно до внедрения средств защиты. Такой подход является превентивной мерой обеспечения надежности и позволяет исключить ухудшение защищенности ИС еще до внедрения в нее СЗИ, а также организовать регрессионное тестирование самих средств защиты.

Предложенный алгоритм верификации программно-аппаратных СЗИ позволяет оценить критичность ошибок, возникающих при выполнении вычислимых по критериям из раздела 2.2 функций безопасности, провести анализ полученных результатов и степени влияния ошибок на защищенность системы. При этом на основе проведенного анализа с использованием положений теории оптимизации и принятия решений принимается решение об успешном завершении

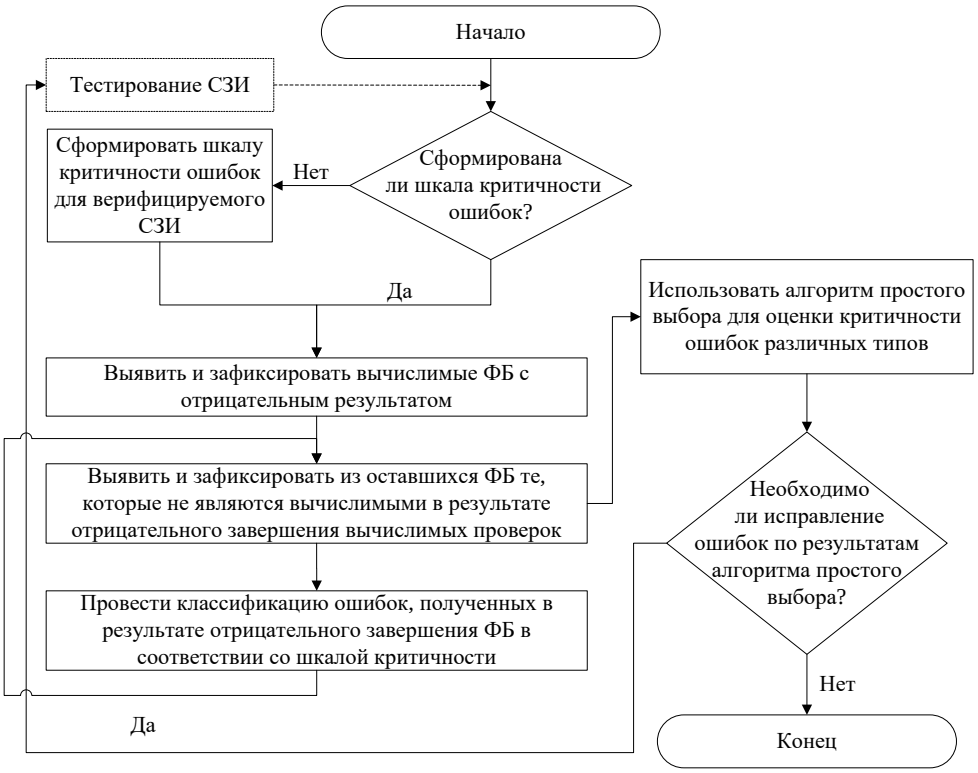


Рисунок 2.14 – Блок-схема алгоритма верификации программно-аппаратных СЗИ, реализующих функции безопасности, основанный на классификации обнаруженных ошибок, анализе степени их критичности и влияния на защищенность системы или данных

тестирования или необходимости исправления выявленных ошибок.

2.5. Способ тестирования функций безопасности программно-аппаратных средств защиты информации, учитывающий состояния аппаратной компоненты

Для применения существующих способов тестирования программного обеспечения к функциям безопасности программно-аппаратных СЗИ необходимо выполнение предложенных в разделе 2.2 общих и частных критериев возможности проведения ручного, автоматического и автоматизированного тестирования. Если условия этих критериев выполняются, то в соответствии с Утверждением 1 и Следствием 1 можно применять предложенный в разделе 2.3 алгоритм решения задачи тестирования программно-аппаратных СЗИ с использованием теории графов для обеспечения полноты и оптимальности этого тестирования. После решения задачи тестирования можно применять предложенный в разделе 2.4 алгоритм верификации программно-аппаратных СЗИ для принятия решения об успешном завершении тестирования или необходимости исправления выявленных ошибок. Основываясь на этом можно предложить способ тестирования функций безопасности программно-аппаратных СЗИ, учитывающий состояния аппаратной компоненты, в соответствии с которым необходимо:

1. Определить по критериям раздела 2.2 принципиаальную возможность проведения тестирования программно-аппаратного СЗИ. При нарушении одного из условий данных критериев – прекратить тестирование и отправить СЗИ на доработку.

2. Провести тесирование с применением предложенного в разделе 2.3 алгоритма решения задачи тестирования программно-аппаратного СЗИ. Если задача тестирования не может быть решена – прекратить тестирование и отправить СЗИ на доработку, так как невозможно обеспечить полноту и оптимальность тестирования его функций безопасности.

3. Провести верификацию с применением предложенного в разделе 2.4 алгоритма верификации программно-аппаратного СЗИ по результатам тестирования. Если в процессе верификации принято решение о необходимости исправления выявленных ошибок – отправить СЗИ на доработку.

Схема предложенного способа тестирования функций безопасности программно-аппаратных СЗИ приведена на Рисунке 2.15.

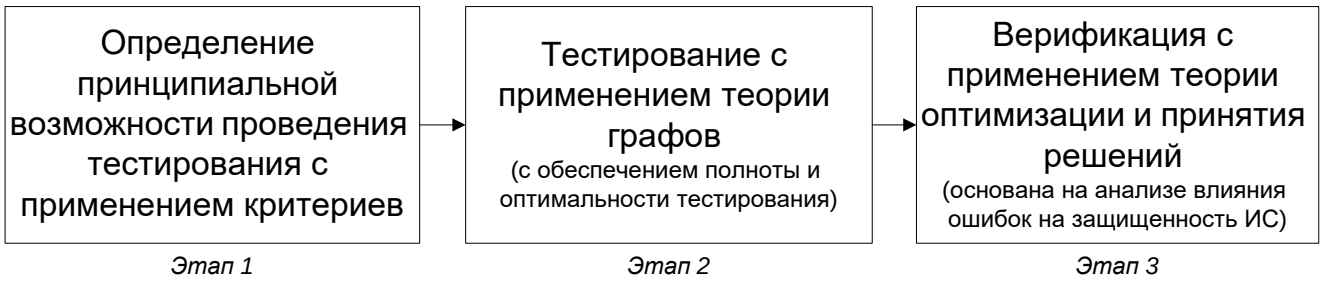


Рисунок 2.15 – Схема способа тестирования функций безопасности программно-аппаратных СЗИ

Если на одном из этапов предложенного способа тестирования программно-аппаратное СЗИ отправляется на доработку, то необходимо определить причину нарушения условий этого этапа: либо функции безопасности по каким-то причинам не являются вычислимыми (например, функции шифрования не является вычислимой до момента генерации криптографических ключей ни ручным, ни автоматическим способом), либо отсутствуют функции перехода в состояния, в которых эти функции становятся вычислимыми (например, без использования специальных средств нельзя выполнить переподключение аппаратной компоненты в СВТ автоматическим способом). На основании этого можно определить, что именно необходимо сделать, чтобы эти условия выполнялись и тем самым можно было применять соответствующий способ тестирования.

Для различных видов функций безопасности программно-аппаратных СЗИ могут нарушатся условия работы среды функционирования средств тестирования или отсутствовать возможность фиксации результатов тестирования. Для выполнения этих условий может потребоваться соответствующим образом либо изменять средства тестирования, либо само средство защиты. Эти действия требуют проведения дополнительных исследований. Помимо этого при использовании способа автоматического тестирования для различных видов функций безопасности программно-аппаратных СЗИ необходимо разрабатывать программы тестирования, часто уникальные для каждого конкретного средства защиты.

Кроме того для функций безопасности программно-аппаратных СЗИ, реализованных на базе мобильной аппаратной компоненты, может отсутствовать возможность подключения/переподключения этой компоненты в процессе настройки или использования средства защиты. Для выполнения этого условия могут потребоваться вспомогательные средства тестирования, требования к которым можно сформировать на основании разработанных критериев. Аналогично для функций безопасности программно-аппаратных СЗИ, реализованных на базе стационарной аппаратной компоненты, должна существовать возможность переноса этой компоненты из одного СВТ в другое. В этом случае также могут потребоваться вспомогательные средства тестирования.

Также для функций безопасности программно-аппаратных СЗИ могут отсутствовать необходимые условия для функционирования реализующей их аппаратной компоненты. Поэтому либо имеет место несовместимость с аппаратной платформой СВТ, либо существует сложность перенаправления аппаратной компоненты в среду функционирования СЗИ или вызванное этим нарушение ее работоспособности.

При этом в ходе тестирования программно-аппаратных СЗИ могут применяться как вспомогательные средства, так и средства тестирования, перечисленные в разделе 1.4, для корректного применения которых необходимо провести исследования и предложить соответствующие требования.

Перечисленным вопросам, касающимся исключительно средств тестирования, в том числе вспомогательных, будет посвящено несколько разделов следующей главы.

2.6. Выводы

На основании проведенных в главе исследований получены следующие результаты:

1. Предложены описательная и формальная модели программно-аппаратного СЗИ, согласно которым любое программно-аппаратное средство защиты может быть представлено в виде конечного детерминированного автомата – совокупности множеств состояний программной и аппаратной компоненты СЗИ, стимулов (в том числе реализуемых и подлежащих тестированию функций безопасности), реакций и переходов из одного состояния в другое.

2. Сформулированы условия применимости способов тестирования ПО к различным видам функций безопасности программно-аппаратных СЗИ. С использованием данных условий для разработанной автоматной модели предложены общие критерии возможности проведения тестирования функций безопасности программно-аппаратных СЗИ. Сформулированы частные критерии для различных видов этих функций безопасности. Данные критерии позволяют проверить принципиальную возможность тестирования программно-аппаратного СЗИ.

3. Разработан алгоритм решения задачи тестирования программно-аппаратных СЗИ, основанный на известных положениях теории графов, и позволяющий обеспечить полноту и оптимальность тестирования.

4. Разработан алгоритм верификации по результатам тестирования программно-аппаратных СЗИ, рассматривающий критичность ошибок не в части нарушения работоспособности объ-

екта тестирования, а с точки зрения возможности нарушения защищенности ИС или данных при некорректной работе функций безопасности. Вводится новый уровень иерархии критериев оценки – критичность самих функций безопасности. При этом критичность каждой ошибки учитывает взаимозависимость вычислимости функций безопасности друг от друга и складывается из совокупности уровней критичности самой ошибки и функции безопасности, в которой она обнаружена.

5. Предложен способ тестирования функций безопасности программно-аппаратных СЗИ, учитывающий состояния аппаратной компоненты, применимый для различных видов программно-аппаратных средств защиты информации, который формирует порядок использования критериев применимости существующих способов тестирования для таких СЗИ, алгоритмов тестирования с применением теории графов и верификации с применением теории оптимизации и принятия решений.

3. Разработка программно-аппаратного комплекса для тестирования функций безопасности СЗИ, реализующего предложенный способ тестирования программно-аппаратных СЗИ

В данной главе рассматривается необходимость и особенности проверки совместимости аппаратной и программной компоненты СЗИ с различными аппаратными платформами СВТ. Также даются рекомендации по использованию средств виртуализации в процессе тестирования функций безопасности программно-аппаратных СЗИ. Выполняется разработка требований к средствам тестирования функций безопасности программно-аппаратных СЗИ с использованием полученных в Главе 2 результатов для различных видов функций безопасности. На основании данных требований формируется общий принцип построения вспомогательных средств тестирования и программ тестирования, необходимых для обеспечения возможности проведения тестирования функций безопасности программно-аппаратных СЗИ. После чего демонстрируется реализация данного принципа на конкретном примере.

3.1. Выработка рекомендаций по тестированию функций безопасности программно-аппаратных средств защиты информации на различных аппаратных платформах

При тестировании функций безопасности программно-аппаратных СЗИ могут возникать ошибки совместимости с различными аппаратными платформами, связанные с версиями BIOS/UEFI, прерываниями, питанием, рабочим напряжением и другими характеристиками СВТ (см. разделы 1.2 и 1.3). Такие ошибки характерны для функций безопасности программно-аппаратных СЗИ, взаимодействующих с ОС СВТ, а также не взаимодействующих с ОС и выполняющихся в составе СВТ. Наиболее часто возникновение таких ошибок наблюдается для СЗИ, функции безопасности которых реализованы на базе стационарной аппаратной компоненты.

Для функций безопасности мобильных аппаратных компонент не существует ограничений на аппаратные характеристики СВТ и достаточно соответствия спецификации определенной версии интерфейса. Например, для USB-устройств – спецификации Universal Serial Bus Revision x.x Specification. Однако иногда все равно может существовать необходимость проведения тестирования на разнообразных аппаратных платформах для проверки соответствия интерфейсов СВТ той или иной спецификации. На практике такие несоответствия встречаются достаточно редко и в рассматриваемом случае не требуется проводить масштабных проверок совместимости с множеством СВТ.

Наибольшую зависимость от аппаратной платформы имеют СЗИ, функции безопасности которых реализованы на базе аппаратной компоненты с PCI / PCI-express интерфейсом подключения. Другие интерфейсы для реализации стационарных аппаратных компонент являются менее распространенными, однако имеют аналогичные особенности взаимодействия с аппаратной платформой. В связи с этим не ограничивая общности будем рассматривать именно PCI / PCI-express-устройства.

Для гарантии работоспособности функций безопасности таких СЗИ необходимо проводить тестирование на большом количестве разнообразных конфигураций оборудования, что при сегодняшнем многообразии СВТ является труднореализуемой задачей. При этом тестирование таких СЗИ на определенном количестве выбранных СВТ не позволяет судить о корректности работы СЗИ вообще на любом оборудовании, которое может использовать конечный пользователь [40]. Для того, чтобы гарантировать работоспособность СЗИ на конкретном оборудовании, производители предлагают пользователям предварительно запросить информацию о совместимости и при возможности предоставить свое оборудование для проверки.

Рассмотрим особенности работы программно-аппаратных СЗИ, функционирование которых зависит от аппаратной платформы СВТ, в которое выполняется его встраивание. К таким СЗИ относятся, в том числе и аппаратные модули доверенной загрузки (например, СЗИ НСД «Аккорд-АМДЗ» и «Соболь»), имеющие PCI / PCI-express интерфейс подключения. Совместимость таких средств защиты с СВТ будем рассматривать с точки зрения совместимости их аппаратной и программной компоненты с аппаратной платформой СВТ.

Под совместимостью аппаратной компоненты, реализующей функции безопасности СЗИ, с СВТ будем понимать ее принципиальную способность функционировать на различных СВТ, например, как PCI / PCI-express-устройство. А под совместимостью программной компоненты – способность внутреннего ПО выполнять свои функции безопасности на различных СВТ.

Средства защиты с PCI / PCI-express интерфейсом подключения, как правило, поддерживают работу на IBM PC-совместимых СВТ [70], которые, в свою очередь, должны соответствовать определенному набору стандартов:

– архитектура микропроцессора – x86 (IA-32) или x86-64 (AMD64) [87, 103];

– интерфейс подключения – PCI и/или PCI-express (стандарты PCI 2.x-3.x и PCI-express 1.x-3.x) [20, 67];

– встроенное ПО типа PC BIOS (в соответствии со стандартами BIOS Boot Specification, Plug and Play BIOS Specification, BIOS Enhanced Disk Drive Specification и другими) или UEFI (стандарт Unified Extensible Firmware Interface Specification версий 2.0 и выше) [14, 114].

При соблюдении данных стандартов производителями СВТ можно гарантировать совместимость СЗИ, использующих соответствующие интерфейсы подключения для своей аппаратной компоненты, на всем спектре таких СВТ. Однако практика показывает, что существует множество исключений – некоторые производители СВТ не полностью соблюдают данные стандарты [84].

Существует два уровня возможной несовместимости аппаратной компоненты с интерфейсом подключения: физический (разъем, размеры, питание и временные параметры сигналов); логический (логика работы интерфейса, ошибка конфигурации).

На физическом уровне при реализации интерфейса PCI / PCI-express существуют допустимые по стандарту диапазоны и отклонения, зависящие от параметров микросхем или особенностей разводки печатной платы. На конкретно взятом СВТ происходит сложение этих погрешностей, что может приводить к несовместимости с программно-аппаратными СЗИ и, тем самым, к нарушению возможности выполнения функций безопасности или стимулов автомата.

Помимо этого, некоторые производители СВТ могут не соблюдать стандарты, например, на соответствующей шине может отсутствовать необходимое напряжение [84]. Это также приводит к нарушению возможности выполнения некоторых функций безопасности и стимулов.

Наиболее часто аппаратная несовместимость модулей доверенной загрузки с СВТ встречается в ноутбуках, в ряде из которых имеется только один разъем Mini PCI-express, при этом иногда он имеет формат half size. Данный разъем, как правило, используется для подключения модуля Wi-Fi, поэтому для возможности применения АМДЗ необходимо извлечь данный модуль и установить в разъем аппаратную компоненту СЗИ. Однако даже в этом случае не гарантируется совместимость СЗИ с СВТ, так как многие производители, например, для ускорения загрузки СВТ, игнорируют инициализацию устройства, подключенного к этому разъему. При этом может игнорироваться и наличие расширения BIOS.

Также некоторые производители ноутбуков и моноблоков применительно к интерфейсу PCI используют список разрешенных устройств – «белый» список (англ. White List). При подключении PCI-устройства к СВТ, не включенного в White List, загрузка СВТ блокируется еще на уровне BIOS/UEFI.

Выделим основные направления, в которых необходимо обеспечивать совместимость программной компоненты модулей доверенной загрузки с СВТ [84]: принцип перехвата управления; совместная работа встроенной ОС СЗИ и его программной компоненты; принцип продолжения загрузки ОС СВТ.

Перехват управления момента загрузки ОС СВТ происходит в два этапа. Сначала непосредственно после самого BIOS СЗИ инициализируется как его расширение и устанавливает собственный обработчик прерывания чтения загрузочного сектора с какого-либо носителя информации, например, с жесткого диска. Затем при загрузке СВТ с носителя информации выполняется передача управления СЗИ. Данный принцип перехвата является универсальным для BIOS СВТ. При этом существует ряд реализаций нестандартных BIOS, для которых этот принцип нарушается, например, из-за затирания отладочных регистров процессора и невозможности передачи управления в собственном обработчике прерываний. Также описанный принцип перехвата не работоспособен на СВТ с UEFI. Совместимость АМДЗ с такими СВТ достигается за счет использования модуля обратной совместимости (англ. Compatibility Support Module, CSM), который обеспечивает процесс загрузки ОС в режиме Legacy.

Ошибки с UEFI могут возникать при совместной работе встроенной ОС СЗИ и его программной компоненты, так как некоторые исполнения ПО аппаратных модулей доверенной загрузки основаны на вызовах сервисов BIOS (ввод информации, вывод изображения и чтение с носителей информации). В таких случаях корректная совместная работа возможна при использовании модуля CSM. Ряд исполнений ПО рассматриваемых средств защиты могут иметь ошибки совместимости сборки его внутренней ОС с многообразием существующих СВТ. К таким ошибкам относятся невозможность запуска ядра или графической оболочки, корректная работа с дисковой подсистемой СВТ. Для исключения перечисленных ошибок ОС СЗИ поддерживает минимально необходимый набор драйверов. А это значит, что отключена поддержка многопроцессорности, используются универсальные драйверы для поддержки видео, дисковой

подсистемы и так далее. Однако некоторые СВТ требуют поддержки специализированных драйверов, например, для RAID-контроллеров. В связи с этим использование стандартной ОС СЗИ с такими СВТ невозможно и требует доработки его программной компоненты.

Принцип продолжения загрузки ОС СВТ в АМДЗ основан на использовании сервисов BIOS СВТ. При этом могут возникать ошибки, связанные с нарушением функционирования задействованных сервисов BIOS после работы полноценной ОС СЗИ со своими драйверами устройств (например, USB-стека и дисковой подсистемы). В частности, с этим связана невозможность загрузки с USB-устройств и загрузки с некоторых типов RAID-контроллеров после работы СЗИ. Использование вместо сервисов BIOS возможностей UEFI позволяет избежать описанных сложностей принципа продолжения загрузки ОС СВТ после СЗИ [84].

Таким образом, для корректной работы функций безопасности программно-аппаратных СЗИ на различных СВТ необходимо обеспечить совместимость его аппаратной и программной компоненты с аппаратной платформой. Для функций безопасности, выполняющихся независимо от СВТ, проверяется корректность взаимодействия программной и аппаратной компоненты программно-аппаратного СЗИ между собой. Для этого требуется проводить тестирование СЗИ на различных СВТ с целью выявления возможных особенностей СВТ и их несоответствия определенным стандартам. При этом в процессе тестирования могут возникать как указанные типовые ошибки (например, на СВТ используется неподдерживаемый RAID-контроллер, на интерфейс подключения подается пониженное напряжение и так далее), так и уникальные для конкретной конфигурации СВТ.

Выше перечислены наиболее распространенные случаи несовместимости программно-аппаратных СЗИ с различными аппаратными платформами, которые могут быть характерны для одного вида средств защиты и не характерны – для другого. Однако в любом случае несовместимость зависит не только от самих программно-аппаратных СЗИ, но и от СВТ, в которых используются функции безопасности средств защиты. При этом проверить совместимость даже одного СЗИ со всеми существующими на сегодняшний момент СВТ, достаточно сложно, так как в каждое СВТ требуется встраивать средство защиты и диагностировать возникающие ошибки, которые могут быть не типовыми. Поэтому одним из решений этого вопроса является организация взаимодействия производителей СЗИ и СВТ в целях проверки совместимости оборудования. Вследствие большого количества необходимых проверок дополнительно требуется автоматизировать процесс тестирования средств защиты на разных аппаратных платформах с использованием результатов Главы 2, а также использовать соответствующие программы тестирования и средства виртуализации или вспомогательные средства, позволяющие сделать вычислимыми функции безопасности и стимулы таких СЗИ.

3.2. Выработка рекомендаций по применению средств виртуализации при тестировании функций безопасности программно-аппаратных средств защиты информации

Как было показано в предыдущих разделах, средства виртуализации можно использовать без каких-либо ограничений для тестирования программных СЗИ. Однако применение таких средств может быть затруднено для программно-аппаратных СЗИ из-за необходимости перена-

правления реализующей функции безопасности аппаратной компоненты в виртуальную среду. Для функций безопасности программно-аппаратных СЗИ, взаимодействующих со средой ОС СВТ, применение средств виртуализации наименьшим образом влияет на результаты тестирования. А для функций безопасности программно-аппаратных СЗИ, не взаимодействующих с ОС СВТ и выполняющихся независимо от СВТ, применение средств виртуализации невозможно в принципе, так как такие средства защиты представляют собой отдельные модули, которые могут взаимодействовать со средой виртуализации только по сети.

В процессе тестирования программно-аппаратных СЗИ использование средств виртуализации может потребоваться, например, в следующих ситуациях [28]: для эмуляции большого количества пользователей, процессов и сеансов подключений (при нагрузочном тестировании); при проведении тестирования на большом количестве программных платформ (в различных версиях и разрядностях ОС); для последовательного тестирования на нескольких СВТ (с переподключением аппаратной компоненты СЗИ); при тестировании СЗИ, часть функций которого выполняются до старта сессии пользователя в ОС или до загрузки ОС СЗИ.

Для программно-аппаратных СЗИ, функции безопасности которых выполняются независимо от СВТ, единственным вариантом применения средств виртуализации в целях тестирования является первая указанная выше ситуация. При этом средства виртуализации используются не для самих программно-аппаратных СЗИ, а для ОС, которые взаимодействуют с ними по сети.

Средства виртуализации упрощают масштабируемость и позволяют автоматизировать ряд проверок, что невозможно сделать на физических СВТ. При этом возможно ухудшение «чистоты» эксперимента посредством ввода еще одного уровня – гипервизора виртуальной среды. На данном уровне могут возникать ошибки, не связанные с функционированием самого СЗИ или аппаратных составляющих СВТ. Также необходимо учитывать, что эмулируемые с использованием средств виртуализации аппаратные платформы могут не соответствовать аналогичным физическим СВТ. В таких случаях работоспособность СЗИ в виртуальной среде не гарантирует его работоспособности на реальном СВТ с аналогичными характеристиками. Например, если используемые интерфейсы подключения СЗИ к физическому СВТ не соответствуют спецификации.

Однако некоторые среды виртуализации не поддерживают перенаправление в ВМ всех существующих аппаратных компонент СЗИ. При использовании виртуализации меньше всего сложностей возникает с программно-аппаратными СЗИ, имеющими мобильную аппаратную компоненту, реализующую функции безопасности, взаимодействующие со средой ОС СВТ. Средства виртуализации, как правило, поддерживают перенаправление таких устройств в ВМ и существуют соответствующие драйверы для поддержки виртуальных интерфейсов подключения в ОС ВМ. В процессе применения средств виртуализации для тестирования программно-аппаратных СЗИ, функции безопасности которых не взаимодействуют с ОС и выполняются в составе СВТ, могут возникать сложности, например, связанные с возможностью инициации загрузки с СЗИ или перехватом управления до загрузки ОС ВМ. Однако в тех средствах виртуализации, в которых используются программные реализации BIOS (и других составляющих СВТ), аналогичные аппаратным и соответствующие спецификации, но не взаимодействующие напря-

мую с аппаратными средствами физического СВТ, таких сложностей не возникает. Например, в KVM используется SeaBIOS, соответствующий спецификации [14, 114].

При использовании средств виртуализации для тестирования функций безопасности, реализованных на базе стационарной аппаратной компоненты, необходимо выполнять ее перенаправление в среду виртуализации при помощи следующих технологий – AMD I/O Virtualization Technology (AMD IOMMU) [98] или Intel Virtualization Technology for Directed I/O (Intel VT-d) [102]. В основе этих технологий лежит применение специального устройства управления памятью ввода-вывода (англ. input/output memory management unit, IOMMU), позволяющего напрямую использовать в ВМ различные периферийные устройства через таблицы отображения прерываний и прямого доступа к памяти (англ. Direct Memory Access, DMA). К таким устройствам относятся, например, контроллеры с интерфейсом PCI / PCI-express. При этом поддержка интерфейса Mini PCI-express возможна с использованием переходника Mini PCI-express – PCI-express. Для корректной работы технологий AMD IOMMU или Intel VT-d они должны поддерживаться процессором, материнской платой, системным/внутренним ПО СВТ (BIOS или UEFI) и ОС СВТ, в которой используется средство виртуализации. В случае отсутствия поддержки этих технологий в одном из указанных компонент СВТ, перенаправление аппаратной компоненты, реализующей функции безопасности, в ВМ будет невозможно.

Таким образом, при тестировании функций безопасности программно-аппаратных СЗИ, реализованных на базе стационарной аппаратной компоненты (в том числе с PCI-интерфейсом подключения), с помощью средств виртуализации необходимо использовать специализированные СВТ, в которых все компоненты поддерживают технологию AMD IOMMU или Intel VT-d. Для обеспечения возможности перенаправления таких устройств в ВМ необходимо также обеспечить поддержку этих технологий в ОС. Например, в ОС Linux и среде виртуализации KVM (Kernel-based Virtual Machine) для перенаправления PCI-устройств в ВМ с помощью технологии Intel VT-d, необходимо использовать ядро ОС со следующими параметрами конфигурации [28, 107]:

– CONFIG_PCI_STUB или CONFIG_VFIO и CONFIG_VFIO_PCI – для обеспечения возможности «отвязки» PCI-устройств от драйверов ОС и их перенаправления в среду ВМ;

– CONFIG_INTEL_IOMMU и CONFIG_INTEL_IOMMU_SVM или CONFIG_IOMMU_SUPPORT – для включения поддержки Intel VT-d с использованием таблиц отображения DMA (для трансляции всех обращений к физической памяти из ВМ);

– CONFIG_IRQ_REMAP – для поддержки отображения прерываний.

В дальнейшем для перенаправления PCI-устройств в среду виртуализации KVM, перед запуском ВМ необходимо сначала открепить устройство от драйвера ОС. Затем требуется прикрепить PCI-устройство к драйверу, который будет использоваться для его перенаправления в среду виртуализации (pci_stub или vfio-pci). После этого можно запускать ВМ, передавая в качестве параметра расположение устройства на шине PCI. Например, «-device pci-assign,host=03:00.0,id=amdz0», где 03:00.0 – интерфейс, в который подключено PCI-устройство [107]. Пример загрузки и перехвата управления после BIOS ВМ с использованием СЗИ НСД «Аккорд-АМДЗ» приведен на Рисунке 3.1.

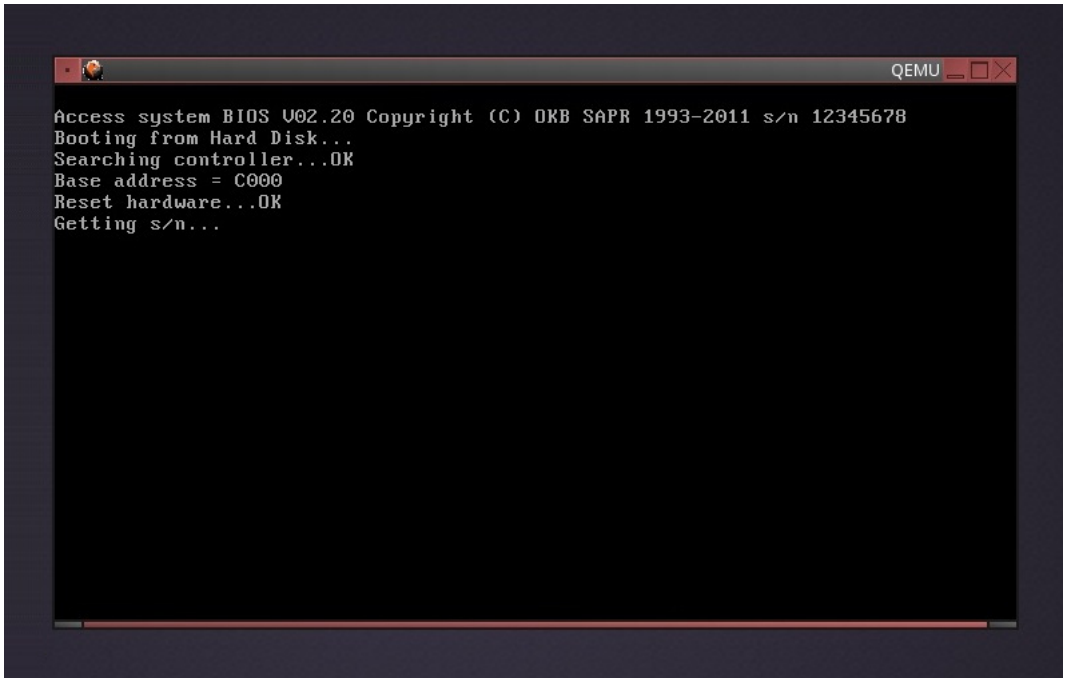


Рисунок 3.1 – Загрузка АМДЗ в виртуальной среде, перехват управления после BIOS ВМ

В случае необходимости подключения аппаратного идентификатора к USB-порту СВТ, а не к самому СЗИ, для успешного прохождения процедуры идентификации АМДЗ в ВМ необходимо также выполнить его перенаправление в ОС виртуальной машины.

Необходимо отметить, что при таком перенаправлении аппаратной компоненты СЗИ будет контролировать целостность BIOS, программных и «технических» средств ВМ, а не реального СВТ, в котором эта ВМ запущена [28].

Как и при работе АМДЗ на реальном СВТ, после успешного завершения процедур идентификации и аутентификации, КЦ программных и «технических» средств ВМ, управление передается загрузчику ОС, при этом для Администратора доступен выбор источника загрузки (см. Рисунок 3.2).

Также использование средств виртуализации позволяет упростить процесс тестирования функций безопасности программно-аппаратных СЗИ на нескольких СВТ. Особенно это касается способа автоматического тестирования. При этом тестирование проводится на одном СВТ, на котором доступны и запускаются все ВМ, к которым поочередно подключается аппаратная компонента СЗИ.

Кроме того, средства виртуализации можно использовать для тестирования программно-аппаратного СЗИ, часть функций безопасности которого выполняется до старта сессии пользователя в ОС СВТ (например, идентификация и аутентификация) или до загрузки ОС СЗИ. При этом в обоих случаях тестирование проводится в ОС СВТ с установленной средой виртуализации, а указанные функции безопасности выполняются в ВМ. Помимо этого тестирование дополнительно выполняется в ОС ВМ (или ОС СЗИ), из-за того что в ОС СВТ, в которой запущена ВМ, не всегда возможно определить результат завершения таких функций безопасности.

В результате использования средств виртуализации в процессе тестирования функций безопасности программно-аппаратных СЗИ, реализованных как на базе мобильной, так и на базе

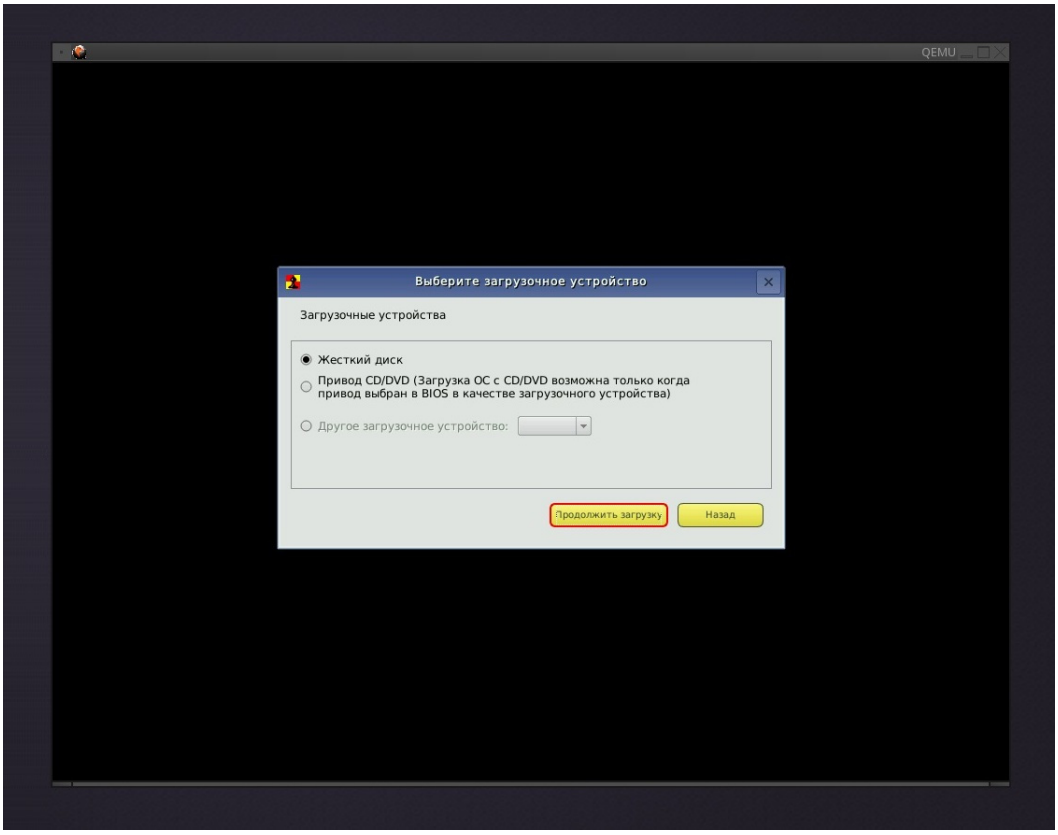


Рисунок 3.2 – Окончание работы АМДЗ, выбор источников загрузки ВМ

стационарной аппаратной компоненты, возможно, и позволяет решить ряд задач. Например, выполнять автоматическое тестирование на нескольких СВТ, тестировать функции безопасности, выполняющиеся до старта сессии пользователя в ОС СВТ или до загрузки ОС СЗИ. При этом для тестирования функций безопасности программно-аппаратных СЗИ, реализованных на базе мобильной аппаратной компоненты, достаточно использовать только встроенные возможности средств виртуализации по их перенаправлению. А для средств защиты со стационарными аппаратными компонентами (PCI / PCI-express / Mini PCI-express или подобных, более современных интерфейсов на материнской плате СВТ), необходимо применять специализированные СВТ, в которых процессор, материнская плата, системное/ внутреннее ПО (BIOS и UEFI) и ОС поддерживают технологии AMD IOMMU или Intel VT-d, а используемое средство виртуализации должно позволять осуществлять перенаправление компонент такого типа.

На основании вышесказанного можно предложить алгоритм тестирования функций безопасности программно-аппаратных СЗИ с использованием средств виртуализации. Этот алгоритм применим для всех программно-аппаратных СЗИ, кроме тех, чьи функции безопасности выполняются независимо от СВТ. Для таких СЗИ средства виртуализации достаточно использовать при взаимодействии по сети с целью эмуляции большого количества пользователей, процессов и сеансов подключений. Предложенный алгоритм позволяет сделать вычислимыми функции безопасности и стимулы, не являющиеся вычислимыми (на основе результатов из Главы 2) на физических СВТ, и заключается в следующем [107]:

1. Необходимо использовать средства виртуализации, выполняя с помощью их стандартных возможностей перенаправление в ВМ аппаратной компоненты СЗИ, реализующей функции

безопасности. При этом программная реализации интерфейса подключения средства защиты, а также компонент ВМ, соответствующих компонентам физических СВТ, должна соответствовать существующим стандартам и спецификациям.

2. Для тестирования функций безопасности, реализованных на базе стационарной аппаратной компоненты, необходимо применять специализированные СВТ, в которых все компоненты (процессор, материнская плата, BIOS или UEFI, ОС) поддерживают технологию AMD IOMMU или Intel VT-d, а используемое средство виртуализации должно позволять осуществлять перенаправление компонент такого типа.

3. При тестировании в ОС ВМ функций безопасности, реализованных на базе мобильной аппаратной компоненты, в случае необходимости выполнять ее переподключение с использованием стандартных возможностей средств виртуализации.

4. При тестировании функций безопасности, выполняющихся до старта сессии пользователя в ОС СВТ или до загрузки ОС СЗИ необходимо проводить тестирование (для способа ручного тестирования – ручные проверки, для способа автоматического тестирования – запуск программ тестирования) и в ОС СВТ и в ОС ВМ (ОС СЗИ).

5. При необходимости тестирования функций безопасности на нескольких СВТ использовать такое же количество ВМ с поочередным подключением аппаратной компоненты.

Блок-схема предложенного алгоритма тестирования приведена на Рисунке 3.3 (АК – аппаратная компонента СЗИ, СВ – средство виртуализации).

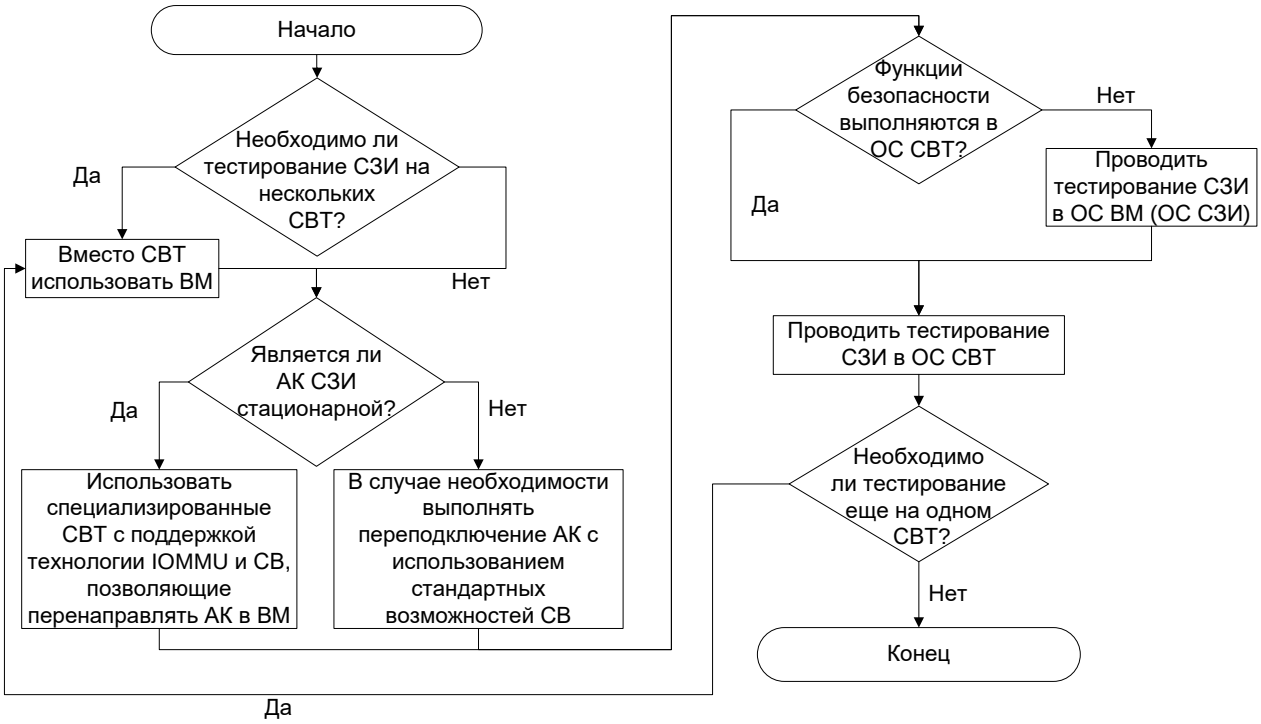


Рисунок 3.3 – Блок-схема алгоритма тестирования функций безопасности программно-аппаратных СЗИ с использованием средств виртуализации

Таким образом, в разделе предложен алгоритм тестирования функций безопасности программно-аппаратных СЗИ путем использования средств виртуализации в тех случаях, когда какие-либо функции безопасности или стимулы не являются вычислимыми согласно результатам

из Главы 2 при тестировании на реальных аппаратных платформах. Применение же виртуальных машин может позволить сделать эти функции и стимулы вычислимыми. При предъявлении повышенных требований к «чистоте» эксперимента следует использовать вспомогательные средства тестирования, также позволяющие добиться вычислимости функций безопасности и стимулов программно-аппаратных СЗИ. Такие средства будут рассмотрены в одном из следующих разделов.

3.3. Обоснование требований к средствам тестирования функций безопасности программно-аппаратных средств защиты информации

Сформируем требования к средствам тестирования функций безопасности программно-аппаратных СЗИ, проанализировав результаты Главы 2. В случае с ручным тестированием все необходимые для проверки действия выполняются тестировщиками вручную зачастую без использования каких-либо средств. Однако возможно применение статических анализаторов, фаззеров и тому подобное, то есть средств тестирования, к которым необходимо сформулировать требования. Для автоматического тестирования помимо уже перечисленных средств необходимо применение также средств автоматизации и реализованных с их помощью программ тестирования, которые позволяют проводить тестирование функций безопасности программно-аппаратных СЗИ автоматическим способом с или без использования вспомогательных средств.

В соответствии с положениями Главы 2 предполагается, что программно-аппаратное СЗИ до начала тестирования должно находится в начальном состоянии v_0, а это значит, что оно изначально находится в этом состоянии, либо тестировщик вручную или программы тестирования автоматически должны выполнить перевод СЗИ в это состояние.

Помимо этого при действиях тестировщика, либо в программах тестирования в соответствии с теми же результатами Главы 2 должны быть реализованы функции перехода в состояния с потенциально вычислимыми функциями безопасности. Например, для функции шифрования в программе тестирования должна быть реализована генерация ключей, которая в данном случае является требуемой функцией перехода, позволяющей проверить корректность шифрования.

Для корректного выполнения каждой последующей проверки необходимо, чтобы после выполнения предыдущей СЗИ находилось в нужном состоянии, иначе результаты тестирования либо будут некорректными, либо оно вообще не сможет быть проведено и такая проверка завершится с ошибкой. Перевод СЗИ в нужное состояние должен быть выполнен или тестировщиком вручную, или быть реализован в программе тестирования для выполнения автоматически, например, путем вызова нецелевой функции форматирования аппаратной компоненты СЗИ. Поэтому в случае наличия такой необходимости после завершения каждой из проверок должен осуществляться либо переход СЗИ в другое состояние, требуемое для выполнения следующей проверки, либо возврат в состояние, в котором это средство защиты находилось до начала выполнения текущей проверки (иначе может быть оказано влияние на выполнение последующих проверок).

Помимо этого, так как некоторые из функций безопасности программно-аппаратного СЗИ не могут быть выполнены в нескольких состояниях, то проверку этих функций и в ручную

и с использованием программ тестирования необходимо проводить во всех таких состояниях. Ели этого не сделать, то будут проверены не все возможные варианты функционирования программно-аппаратного СЗИ, а следовательно, в соответствии с результатами Главы 2 будет нарушена полнота тестирования.

Реализация описанных выше действий позволит осуществлять динамическое тестирование программно-аппаратных СЗИ, при котором начало выполнения следующей ручной проверки или запуск на выполнение следующей автоматической проверки функции безопасности зависит от текущего состояния средства защиты: либо она вычислима и запускается, либо не является вычислимой и необходимо вызвать функцию перехода в соответствующее состояние СЗИ. В таком случае для автоматического тестирования появляется возможность автоматически определять состав проверок, подлежащих выполнению.

В соответствии с построением модели программно-аппаратного СЗИ из Главы 2 при выполнении функции безопасности будет получен некоторый результат выполнения (выход автомата) в определенном состоянии программно-аппаратного СЗИ. В случае отсутствия ошибок автомат переходит в определенное новое состояние, а при их наличии – не переходит в это состояние. Поэтому каждая вычислимая функция безопасности должна возвращать один из результатов завершения проверки (переходить или не переходить в определенное новое состояние). При этом каждая ручная проверка должна завершаться визуальным результатом, а автоматическая – возвращать набор выходных данных, например, журнал работы, код ошибки, дополнительный результат и так далее, позволяющий провести анализ процесса ее выполнения и результата.

Необходимо отметить, что в случае некорректной работы функций безопасности программно-аппаратного СЗИ, может нарушиться вычислимость других функций безопасности, для начала выполнения которых они используются. В данном случае проверки таких функций безопасности должны быть исключены из программ тестирования и ручных проверок, так как заранее известно, что результат выполнения будет отрицательным.

На основании приведенных выше фактов можно определить какой именно функциональностью должны обладать средства тестирования программно-аппаратных СЗИ, разделяемых по виду взаимодействия с защищаемым СВТ в ИС, и какие условия необходимы для их корректного функционирования.

Помимо этого с использованием частных критериев возможности проведения ручного и автоматического тестирования функций безопасности программно-аппаратных СЗИ, реализованных на базе мобильной и стационарной аппаратной компоненты можно предложить требования к средствам тестирования для функций безопасности, разделяемых по виду реализующей аппаратной компоненты. В соответствии частным критерием возможности проведения ручного и автоматического тестирования для функций безопасности программно-аппаратных СЗИ, реализованных на базе мобильной аппаратной компоненты, должна существовать возможность ручного или автоматического подключения и отключения аппаратной компоненты к/от СВТ соответственно. А согласно частному критерию возможности проведения ручного и автоматического тестирования функций безопасности программно-аппаратных СЗИ, для функций безопасности, реализованных на базе стационарной аппаратной компоненты, – возможность ручного или ав-

томатического переноса аппаратной компоненты из одного СВТ в другое. На основании этого можно сформулировать требования к ручным действиям тестировщика и к программам тестирования, при соответствии которым станет возможным ручное и автоматическое тестирования указанных видов функций безопасности программно-аппаратных СЗИ, а сами стимулы для осуществления переходов будут являться вычислимыми.

Подведем итог вышесказанному и сформулируем требования к программам тестирования, средствам автоматизации и другим средствам из раздела 1.4 (статические анализаторы, фаззеры и прочие), которые могут применяться для тестирования различных видов функций безопасности программно-аппаратных СЗИ [29, 104].

Требования к средствам тестирования функций безопасности программно-аппаратных СЗИ и среде их применения:

1. Средства тестирования, как существующие, так и разрабатываемые, должны корректно функционировать в среде (например, ОС СВТ или ОС СЗИ), в которой выполняются функции безопасности и осуществляется тестирование. Также должна существовать возможность фиксации результатов тестирования, выполненного при помощи этих средств. Для этого может потребоваться изменение используемых существующих средств тестирования и/или самого программно-аппаратного СЗИ (среды их применения).
2. Программы тестирования должны учитывать следующее:
   – перед выполнением проверок СЗИ должно находится в начальном состоянии (v_0);
   – должны быть реализованы переходы СЗИ во все состояния с потенциально вычислимыми функциями безопасности;
   – после завершения проверок должен осуществляться либо переход СЗИ в другое состояние, необходимое для выполнения следующей проверки (в том числе может остаться неизменным), либо возврат в состояние, в котором это средство защиты находилось до начала выполнения текущей проверки;
   – в кажом состоянии программно-аппаратного СЗИ должны проверяться все функции безопасности, потенциально вычислимые в этом состоянии (то есть обеспечивается полнота тестирования и каждая функция безопасности во всех необходимых состояниях СЗИ);
   – каждая вычислимая функция безопасности должна возвращать один из результатов завершения проверки: переходить или не переходить в определенное новое состояние программно-аппаратного СЗИ в случае отсутствия или наличия ошибок соответственно;
   – если проверка какой-либо функции безопасности, являющейся единственным стимулом для перехода в некоторое состояние СЗИ, завершилась с отрицательным результатом, то для всех функций безопасности, потенциально вычислимых в этом состоянии, результат будет отрицательным и проведение проверок для них не требуется;
   – каждая проверка функции безопасности при завершении ее работы должна возвращать набор выходных данных (журнал работы, код ошибки, дополнительный результат и так далее);

– должна существовать возможность подключения и отключения аппаратной компоненты к/от СВТ соответственно для функций безопасности программно-аппаратных СЗИ, реализованных на базе мобильной аппаратной компоненты;

– должна существовать возможность переноса аппаратной компоненты из одного СВТ в другое для функций безопасности программно-аппаратных СЗИ, реализованных на базе стационарной аппаратной компоненты.

При проведении ручного тестирования требования из пункта 2 могут быть выполнены тестировщиком вручную без применения каких-либо дополнительных средств тестирования. А при автоматическом тестировании данные требования должны быть реализованы в разработанных программах тестирования. При этом для выполнения требований, регламентированных видом реализующей функции безопасности аппаратной компоненты СЗИ, с использованием программ тестирования необходимо применение дополнительных средств, так как существующие средства автоматизации не позволяют провести такие действия автоматическим образом. Для автоматического тестирования таких функций безопасности в случае применения предложенного алгоритма тестирования функций безопасности программно-аппаратных СЗИ с использованием средств виртуализации (см. раздел 3.2) можно использовать встроенные возможности средств виртуализации по перенаправлению в ВМ различных видов аппаратных устройств. Однако, если при тестировании предъявляются повышенные требования к «чистоте» эксперимента или применение средств виртуализации для этого СЗИ невозможно, следует использовать вспомогательные средства тестирования.

При этом реализация таких вспомогательных средств для тестирования функций безопасности СЗИ с мобильной аппаратной компонентой может быть построена на общем принципе – эмуляции физического отключения и подключения СЗИ к определенному интерфейсу СВТ. Например, «eToken», «Рутокен», ПСКЗИ ШИПКА и ПАК «Секрет» (см. раздел 1.2) в ходе настройки и эксплуатации подключаются и отключаются от USB-порта СВТ, а, значит, при их тестировании также необходимо эмулировать эти действия. При этом особенности работы вспомогательных средств подключения/отключения СЗИ к СВТ будут зависеть только от интерфейса подключения, то есть такие средства можно реализовать относительно универсальным способом.

Вспомогательные средства для тестирования функций безопасности СЗИ, реализованных на базе стационарной аппаратной компоненты, должны позволять автоматизированно встраивать СЗИ в СВТ. Однако такие программно-аппаратные СЗИ могут встраиваться в СВТ различным образом в зависимости от реализации конкретного средства защиты. Например, встраивание СЗИ НСД «Аккорд-АМДЗ» происходит путем вскрытия корпуса СВТ и установки контроллера в соответствующий интерфейс подключения PCI / PCI-express / Mini PCI-express [70]. А установка АПМДЗ «Соболь» дополнительно требует перевода аппаратной компоненты в рабочий режим – снятия перемычки, установленной на соответствующий разъем платы [71]. Встраивание других СЗИ может также иметь свои нюансы и зависит от конкретной реализации того или иного средства защиты. Таким образом, для таких СЗИ нельзя выделить общие принципы встраивания. Помимо этого осуществить автоматизацию процесса встраивания некоторых СЗИ на практике

может быть затруднительно, например, автоматически вскрыть корпус и подключить АМДЗ в соответствующий интерфейс СВТ (при необходимости тестирования на большом разнообразии аппаратных платформ, см. раздел 3.1) или изменить приоритет загрузки в BIOS возможно только с применением специализированных промышленных автоматизированных комплексов, что является недоступным для большинства компаний-разработчиков СЗИ. Кроме необходимости встраивания могут существовать и другие особенности, специфичные для конкретных программно-аппаратных СЗИ, функции безопасности которых реализованы на базе стационарной аппаратной компоненты. Например, аппаратную компоненту «Аккорд-АМДЗ» в процессе тестирования необходимо переводить из «технологического» режима в рабочий и наоборот. Такой процесс перевода «Аккорд-АМДЗ» требует вскрытия корпуса СВТ и замыкания определенных контактов (переключения микропереключателя или установки джамперов) на плате аппаратной компоненты СЗИ [70]. Эти действия возможно выполнить вручную, но автоматическое их выполнение вызывает определенные затруднения.

Таким образом, на основании результатов исследований из Главы 2 сформулированы требования к средствам тестирования функций безопасности программно-аппаратных СЗИ и среде их применения. При выполнении полученных требований проверка каждой функции безопасности проводится во всех состояниях, в которых она потенциально вычислима, обеспечивая тем самым полноту проводимого тестирования. При этом необходимо отметить, что при выполнении разработанных требований в процессе тестирования будут присутствовать только вычислимые функции безопасности и стимулы, то есть те, которые можно провести и получить либо положительный результат, либо – отрицательный, свидетельствующий о наличии ошибки. После завершения каждой проверки должен осуществляться либо переход СЗИ в другое состояние, требуемое для выполнения следующей проверки, либо возврат в состояние, в котором это средство защиты находилось до начала выполнения текущей, в противном случае может быть оказано влияние на выполнение последующих проверок. Реализация таких действий позволяет осуществлять динамическое тестирование функций безопасности программно-аппаратных СЗИ, при котором начало выполнения следующей проверки функции безопасности зависит от текущего состояния средства защиты: либо она является вычислимой и происходит выполнение следующей функции безопасности, либо – не является вычислимой и необходимо выполнить переход в соответствующее состояние СЗИ. В таком случае для автоматического тестирования появляется возможность автоматически определять состав проверок, подлежащих выполнению. При этом необходимо учитывать, что сформулированные требования к средствам тестирования при автоматическом тестировании могут быть выполнены только путем разработки программ тестирования и применения вспомогательных средств, реализующих подключение/отключение к/от СВТ аппаратной компоненты СЗИ и ее перенос в другое СВТ.

3.4. Обоснование состава вспомогательных средств, используемых в программно-аппаратном комплексе для тестирования функций безопасности программно-аппаратных СЗИ и рекомендаций по их практическому использованию

В данном разделе приводятся обоснование состава вспомогательных средств, используемых в программно-аппаратном комплексе для тестирования функций безопасности программно-аппаратных СЗИ, а также формулирубтся рекомендации по их практической реализации. Затем дается краткое описание вспомогательного средства тестирования, разработанного на основании данных рекомендаций.

Как показано в разделе 3.3 вспомогательное средство тестирования для средств защиты должно автоматическим способом эмулировать подключение и отключение СЗИ к/от физического СВТ. Выполнить такую эмуляцию можно с помощью программных средств – с использованием BIOS, путем настройки конкретной ОС. Однако такие средства могут полностью не эмулировать физическое отключение аппаратной компоненты, например, может не происходить отключение питания. Это затрудняет использование вспомогательных программных средств в процессе тестирования функций безопасности программно-аппаратных СЗИ. Также такие вспомогательные программные средства могут быть неуниверсальными, требующими адаптации для различных версий ОС, BIOS и так далее. В связи с этим для подключения и отключения к СВТ аппаратных компонент СЗИ целесообразно применять именно программно-технические/программно-аппаратные средства тестирования [104]. Это связано с тем, что только устройство с аппаратной частью может полностью эмулировать физическое отключение и подключение СЗИ к определенному интерфейсу, при этом быть достаточно универсальным и не зависеть от версии BIOS, ОС и так далее. Поэтому для программно-аппаратных СЗИ вспомогательное средство тестирования должно представлять собой:

– техническое средство, устанавливаемое между соответствующим интерфейсом СВТ и СЗИ «в разрыв» канала, и осуществляющее его коммутацию в соответствии с командами, поступающими по каналу управления;

– программное обеспечение, позволяющее из ОС СВТ подать соответствующую команду по каналу управления – включить или отключить передачу данных и питание для коммутируемого устройства.

Такое программно-техническое средство по своей сути должно быть коммутатором канала интерфейса подключения аппаратной компоненты СЗИ к СВТ, то есть должно быть способно физически прерывать питание различных подключаемых к нему устройств (например, «eToken», «Рутокен», ПСКЗИ ШИПКА и СН «Секрет», см. раздел 1.2) без их физического отключения или подключения.

При этом данное вспомогательное средство должно быть предназначено для выполнения следующих функций:

– управляемое переподключение устройств в процессе их автоматического тестирования - в этом случае данное вспомогательное средство используется как техническое средство тестирования;

– управляемая блокировка доступа к каналу интерфейса подключения аппаратной компоненты СЗИ к СВТ – как средство защиты информации, реализующее блокировку запрещенных устройств и позволяющее осуществлять доступ к разрешенным устройствам.

Схема подключения СЗИ через вспомогательное программно-техническое средство тестирования к СВТ приведена на Рисунке 3.4 [104].

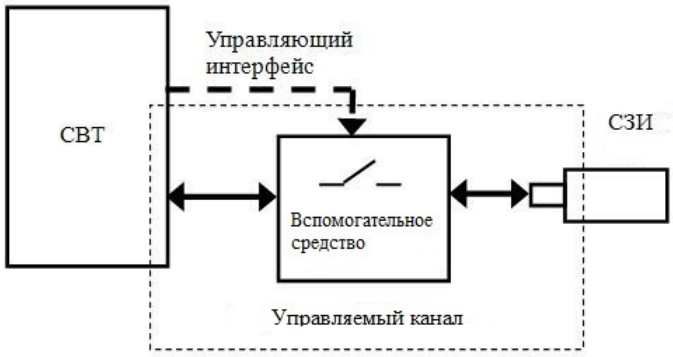


Рисунок 3.4 – Принципиальная схема подключения СЗИ через вспомогательное программно-техническое средство к СВТ

Функционально такое вспомогательное средство должно состоять из следующих компонент [104]:

1. Аппаратная компонента, включающая:
   – микроконтроллер;
   – два разъема, соответствующих интерфейсу подключения аппаратной компоненты СЗИ к СВТ (Далее – I-интерфейс) – I1 и I2, где I1 используется для передачи управляющих команд с СВТ в коммутатор, I2 – для передачи данных в (из) подключаемое к коммутатору устройство;
   – один разъем с I-интерфейсом – I3, используемый для подключения устройств к коммутатору;
   – мультиплексор линий данных между I2-I3;
   – ключ переключения питания I2-I3;
   – средство индикации состояний коммутатора (например, красный и зеленый светодиоды);
   – переключатель сброса контроллера (для осуществления перезагрузки внутреннего ПО);
   – вход внешнего питания микроконтроллера, ключей и светодиодов (питание может подаваться по I1 или от внешнего источника с использованием данного входа).
2. Внутреннее ПО (firmware) – ПО, запускающееся из аппаратной компоненты при поступлении питания и выполняющее основную функциональность программно-аппаратного комплекса.
3. Внешнее ПО, состоящее из:
   – библиотеки для встраивания и использования комплекса в стороннем ПО;

– утилиты командной строки для выполнения коммутации устройств (с использованием библиотеки).

Принцип работы такого средства должен быть следующим. При включении питания аппаратной компоненты (то есть при подключении I1 к СВТ или подаче питания на вход внешнего питания) запускается внутреннее ПО, которое последовательно производит следующие действия [32, 104]:

– тестирует работоспособность аппаратной компоненты;

– оповещает о готовности аппаратной компоненты к работе (например, включает красный светодиод);

– отключает передачу данных и питание на I2-I3;

– оповещает об отключении I2-I3 (например, выключает зеленый светодиод);

– ожидает сообщений по управляющему интерфейсу I1.

С использованием библиотеки для встраивания комплекса в стороннее ПО или работающей на ее основе утилиты командной строки можно подать следующие команды аппаратной компоненте [30, 104]: включить передачу данных и питание на I2-I3 (включение может идентифицироваться, например, свечением зеленого светодиода); отключить передачу данных и питание (может идентифицироваться, например, выключением зеленого светодиода); запросить состояние коммутации.

Коммутация канала (I2-I3) обеспечивается синхронным включением и выключением мультиплексора линий данных и питания. Команды между внешним и внутренним ПО передаются путем обмена сообщениями специального формата по управляющему каналу I1.

Внутреннее ПО при поступлении сообщения (содержащего команду для аппаратной компоненты) по управляющему каналу I1 должно осуществлять следующие действия:

– извлекает из сообщения команду для аппаратной компоненты;

– выполняет команду (в зависимости от команды – включает/отключает передачу данных и питание на I2-I3 и, например, соответствующим образом изменяет состояние зеленого светодиода);

– возвращает по I1 результат выполнения команды.

Для использования данного вспомогательного средства тестирования должен быть предусмотрен программный интерфейс (например, реализованный в библиотеке для встраивания и использования комплекса в стороннем ПО), предоставляющий следующий набор функций [32]:

– *std::list<std::string> US_Enumerate()*;

– *std::string US_ON(std::string Switcher)*;

– *std::string US_OFF(std::string Switcher)*,

где Switcher – строка (из списка, возвращаемого US_Enumerate()), идентифицирующая коммутатор, которому передается команда «Switcher X», где X – номер подключенного к СВТ коммутатора, начиная с «0»; US_Enumerate – функция, осуществляющая поиск и вывод списка найденных и доступных в данный момент коммутаторов (либо пустой список – в случае отсутствия подключенных к СВТ коммутаторов); US_ON – функция, передающая команду на включение передачи

данных и питания устройств, подключенных к коммутатору (в случае корректного завершения работы возвращает «SUCCESS», иначе – генерируется исключение типа std:string с описанием ошибки); US_OFF – функция, передающая команду на отключение передачи данных и питания устройств, подключенных к коммутатору (в случае корректного завершения работы возвращает «SUCCESS», иначе – генерируется исключение типа std:string с описанием ошибки).

При использовании рассматриваемого вспомогательного средства тестирования с помощью утилиты командной строки, например, SwitcherConsole.exe, должно быть возможно передавать на вход следующие параметры [32]:

– list – для вывода списка доступных коммутаторов канала, то есть реализуется выполнение функции US_Enumerate из библиотеки.

– on <switcherName> – для включения передачи данных и питания устройств, подключенных к соответствующему коммутатору, то есть реализуется выполнение функции US_ON из библиотеки встраивания и использования комплекса в стороннем ПО. Без указания <switcherName> команда отправляется на первый доступный коммутатор.

– off <switcherName> – для отключения передачи данных и питания устройств, подключенных к соответствующему коммутатору, то есть реализуется выполнение функции US_OFF из библиотеки встраивания и использования комплекса в стороннем ПО. Без указания <switcherName> команда отправляется на первый доступный коммутатор.

Так как настоящее время основным интерфейсом подключения, на базе которого реализуются большинство мобильных СЗИ, является USB, то применение сформулированных рекомендаций целесообразно выполнять сначала именно для этого интерфейса подключения. Поэтому для подтверждения на практике корректности сформулированных рекомендаций разработан так называемый коммутатор USB-канала [31, 32, 34], который реализует все описанные выше функции и предоставляет перечисленные возможности для мобильных программно-аппаратных СЗИ с USB-интерфейсом подключения.

Коммутатор USB-канала с подключенным к нему коммутируемым программно-аппаратным СЗИ с USB-интерфейсом подключения изображен на Рисунке 3.5, при этом USB1 – это I1; USB2 – I2; USB3 – I3.

Разработанный программно-аппаратный комплекс (коммутатор USB-канала) можно использовать в программах тестирования для функций безопасности программно-аппаратных СЗИ, реализованных на базе мобильной аппаратной компоненты с USB-интерфейсом подключения (в том числе – «eToken», «Рутокен», ПСКЗИ ШИПКА и ПАК «Секрет»). Применение данного коммутатора возможно в тех случаях, когда необходимо автоматически подключать/переподключать СЗИ к СВТ, например, для эмуляции действий пользователя при работе с СЗИ.

Коммутатор USB-канала в соответствии с предложенными выше рекомендациями полностью эмулирует физическое отключение и подключение СЗИ к СВТ как на уровне питания, так и на уровне канала передачи данных. Поэтому данный программно-аппаратный комплекс полностью соответствует требованию к вспомогательным средствам автоматизации тестирования функций безопасности программно-аппаратных СЗИ, реализованных на базе мобильной аппаратной компоненты (см. раздел 3.3). Помимо целевого использования коммутатора USB-канала

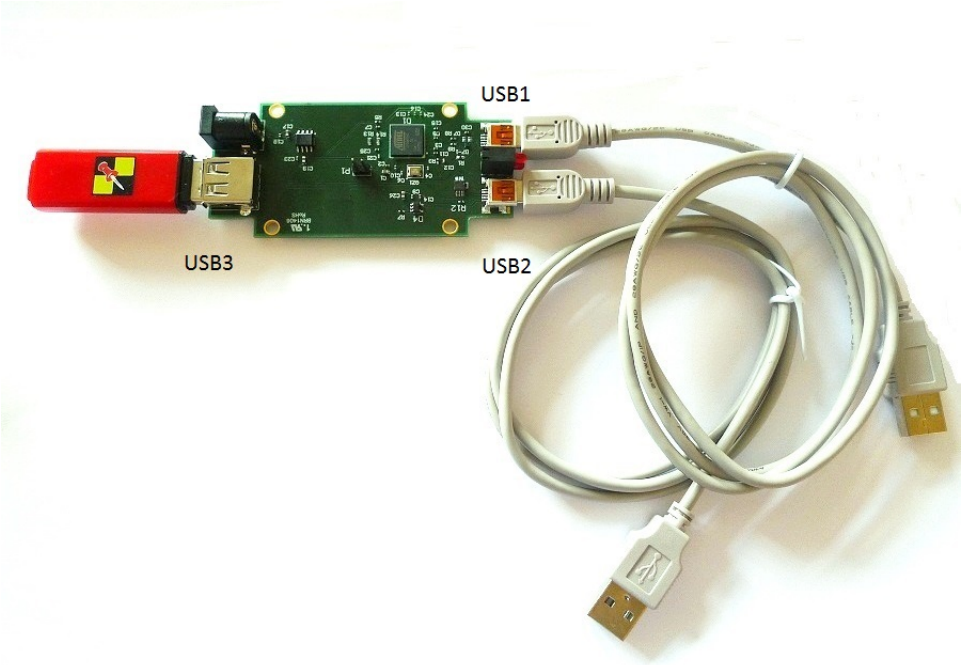


Рисунок 3.5 – Вспомогательное средство тестирования с подключенным коммутируемым программно-аппаратным СЗИ – ПСКЗИ ШИПКА

в качестве программно-технического средства тестирования, возможно его применение и как средства защиты информации для реализации блокировки запрещенных USB-устройств и осуществления доступа к разрешенным устройствам.

При разработке коммутатора USB-канала были проанализированы существующие решения, близкие к нему по выполняемым функциям. Так, в направлениях, не связанных с тестированием средств защиты информации и тестированием вообще, применяются средства, в состав которых входит USB-реле, способное удаленно переключать/выключать USB-устройства (например, в системах типа «умный дом»). Принципиально данное USB-реле может выступать в качестве основы коммутатора USB-канала.

Сравнение возможностей предложенного коммутатора USB-канала с возможным решением – коммутатором, реализованным на базе USB-реле, приведено в Таблице 3.1.

Таблица 3.1 – Сравнение возможностей предложенного коммутатора USB-канала и возможного коммутатора на базе USB-реле

Возможности	**Коммутатор USB-канала**	**Коммутатор на базе USB-реле**
1. Автоматическое подключение/отключение USB-устройств к/от СВТ	+	+
2. Корректировка сигнала	+	- (без специальных доработок)
3. Отсутствие «паразитных» токов (при выключении коммутатора)	+	- (без специальных доработок)

Продолжение таблицы 3.1

Возможности	**Коммутатор USB-канала**	**USB-реле**
4. Возможность адаптации для USB-3.0	+	+ (после осуществления доработок по п.2 и 3)
5. Самодостаточность для эмуляции физического подключения/отключения СЗИ к/от СВТ	+	- (необходимо спроектировать и собрать аналогичную разработанной плату, разработать внутреннее и внешнее ПО)

При использовании USB-реле в качестве основы для коммутатора USB-канала могут возникнуть сложности с корректировкой сигнала или с возникновением «паразитных» токов (при выключении коммутатора), в результате чего поддержка таких интерфейсов, как USB-3.0 может быть затруднена. Для использования USB-реле как основы коммутатора USB-канала необходимо спроектировать и собрать аналогичную плату с устранением описанных недостатков, разработать внутреннее и внешнее ПО. Полностью готовых аналогов разработанному коммутатору USB-канала для тестирования программно-аппаратных СЗИ, в том числе и использующих в своей основе USB-реле, в открытом доступе не найдено.

На основании выше сказанного можно сформулировать рекомендации по практической реализации вспомогательного средства тестирования. Данное средство тестирования должно:

1. Быть программно-техническим, так как только устройство с аппаратной частью может полностью эмулировать физическое отключение/подключение СЗИ к определенному интерфейсу и при этом быть достаточно универсальным и не зависеть от версии BIOS, ОС и так далее.
2. Устанавливаться «в разрыв» канала и осуществлять коммутацию интерфейса СВТ и СЗИ в соответствии с командами, поступающими по каналу управления.
3. Состоять из аппаратной компоненты с внутренним ПО и внешнего программного обеспечения.
4. Включать в свой состав:

– внешнее ПО, позволяющее подать команду по каналу управления – включить или отключить передачу данных и питание для коммутируемого устройства;

– библиотеку для встраивания комплекса в стороннее ПО;

– утилиту командной строки, через которую возможно подавать необходимые команды аппаратной компоненте.

3.5. Рекомендации по практической реализации средств тестирования функций безопасности программно-аппаратных средств защиты информации

В соответствии с разделами 1.3, 2.2 и 3.3 для проведения автоматического тестирования функций безопасности программно-аппаратных СЗИ требуется разработать программы тестирования, руководствуясь перечнем проверок функций безопасности, входными данными из

ПМИ, учитывая при этом необходимость применения в рамках этого процесса вспомогательных программно-технических средств тестирования.

Для каждого конкретного средства защиты программы тестирования являются в большинстве своем уникальными, приближение к универсальности возможно только при разработке программ тестирования для СЗИ, реализующих одни и те же функции безопасности и относящихся к одному и тому же виду: при разделении по степени привязки к СВТ (функционирующие в среде ОС СВТ или независимо от нее) и по виду аппаратной компоненты (мобильная или стационарная). Например, можно реализовать универсальным способом проверки средств криптографической защиты информации различных производителей, функционирующих в ОС СВТ и имеющих мобильную аппаратную компоненту в части тестирования функций генерации ключей и ключевых пар, шифрования и подписи данных. Однако в любом случае нельзя единообразно реализовать проверки всех имеющихся в таких СЗИ функций, так как они могут быть уникальными для конкретного средства защиты. Можно предложить общие принципы разработки программ тестирования для программно-аппаратных СЗИ одного вида (при разделении их по степени привязки к СВТ и по виду аппаратной компоненты) и реализовать универсальные функции проверок для таких средств защиты, а затем дополнить их проверками специфичных функций конкретного средства защиты. При этом необходимо учитывать, что для тестирования различных видов программно-аппаратных СЗИ при их рассмотрении с точки зрения классификации по виду аппаратной компоненты (мобильная и стационарная) необходимо применение различных вспомогательных средств тестирования. А для тестирования средств защиты, взаимодействующих с ОС СВТ и не взаимодействующих с ней необходимо учитывать особенности используемого способа тестирования.

В данном разделе предлагаются несколько различных принципов разработки программ тестирования: для функций безопасности мобильных и стационарных программно-аппаратных СЗИ, функционирующих в ОС СВТ и независимо от нее, на основании которых выполняется разработка программ тестирования для конкретных СЗИ соответствующего вида с использованием предложенного в разделе 2.5 способа тестирования для функций безопасности программно-аппаратных СЗИ и вспомогательного средства тестирования из раздела 3.4. Необходимо отметить, что большинство программно-аппаратных СЗИ относятся либо к мобильным, взаимодействующим с ОС СВТ, либо к стационарным, не взаимодействующим или частично взаимодействующим с данной ОС – некоторые функции безопасности выполняются независимо от ОС СВТ, а некоторые в ней. Поэтому формирование рекомендаций по практической реализации средств тестирования выполняется в первую очередь для таких СЗИ.

Помимо этого в данном разделе выполняется рекомендаций по практической реализации предложенного в разделе 2.4 алгоритма верификации программно-аппаратных СЗИ, на основании которых разработана программа верификации, выполняющая классификацию обнаруженных ошибок, анализ степени их критичности и влияния на защищенность системы.

Рекомендации по практической реализации средств тестирования функций безопасности мобильных программно-аппаратных СЗИ

Разработку программ тестирования для функций безопасности мобильного программно-аппаратного СЗИ, взаимодействующих со средой ОС СВТ, рекомендуется выполнять с применением средства автоматизации TestComplete от SmartBear Software, так как это одно из средств, наиболее подходящих для автоматического тестирования данного вида функций безопасности (см. раздел 1.4).

Для разработки программ тестирования в TestComplete необходимо создать так называемый Project Suite, представляющий собой несколько проектов. Каждый проект состоит из набора программ различного типа, которые можно использовать для автоматического тестирования одной или нескольких составляющих СЗИ.

Существуют следующие типы программ тестирования в TestComplete, рекомендуется использовать один из них:

– *script* – программы тестирования, составляемые вручную разработчиком автоматических тестов на одном из поддерживаемых скриптовых языков (*VBScript*, *JScript*, *C++Script*, *C#Script*, *DelphiScript*). Такая программа описывает последовательность действий автоматического теста, содержит обработчики возникающих событий и вспомогательные функции;

– *keywordTest* – программы тестирования, воспроизводящие «записанную» последовательность действий: щелчок мышью, нажатие клавиш и так далее. Каждому действию сопоставляются ключевые слова (англ. keywords), то есть программы типа KeywordTest являются визуальным представлением аналогичных программ типа Script. Разработка программ тестирования типа KeywordTest не требует навыков программирования, однако их использование имеет ряд ограничений (например, для сложных условий требуется делать ручные вставки кода).

Для упрощения описания последовательности тестирования в программах тестирования TestComplete рекомендуется использовать возможность применения именований (*NameMapping*) и псевдонимов (*Aliases*) для объектов. При этом в NameMapping можно использовать две модели построения списка объектов: в виде дерева (*Tree*) объектов с иерархией или в виде объектов без иерархии (*Flat*). Модель Tree использует более длинные имена (от родительского к дочернему объекту), а модель Flat – более короткие (без промежуточных родительских объектов).

В проект следует добавить перечень тестируемых приложений (*TestedApps*), входящих в тестируемую программную компоненту СЗИ, а также настроить индивидуальные параметры запуска. В программах тестирования к добавленным приложениям можно обращаться с помощью ключевого слова TestedApps.

Для хранения различных дополнительных объектов, например, файлов рисунков с результатами корректного или некорректного завершения работы, в проекте используется хранилище (*Stores*). Такие дополнительные объекты могут использоваться, например, для дальнейшего их сравнения с результатами работы программ тестирования.

Для хранения и отображения перечня результатов с пометкой об успешном или неуспешном выполнении каждого действия используются журналы работы программ тестирования TestSuite Logs/Project Logs. В случае возникновения ошибок или предупреждений в работе программы те-

стирования в журнале указывается место их возникновения – в исходном тексте программы или в описанной последовательности действий, приводится информация о приложении, описание ошибок и предупреждений, включая снимок экрана в моменты их возникновения.

Существуют следующие типы событий в журнале работы программ тестирования:

– ошибки (*errors*), автоматически генерируемые средой автоматизации TestComplete (например, не найден нужный объект, над которым необходимо производить некоторые действия) или описываемые при разработке программ тестирования вручную (например, некорректные результаты определенной проверки);

– предупреждения (*warnings*) – некритичные ошибки, также генерируемые автоматически или описываемые вручную;

– сообщения (*messages*) – информационные сообщения, описываемые при разработке программ тестирования вручную (например, сообщения об окончании проведения той или иной проверки, сообщения для отладки);

– события (*events*) – сообщения, описывающие последовательность действий, выполняемых в процессе работы программы тестирования (например, ввод символов с клавиатуры, щелчок мыши, позиционирование курсора).

Каждое событие в журнале работы программ тестирования сопровождается его описанием с прикреплением следующих дополнительных данных, необходимых для анализа результата выполнения автоматического тестирования:

– изображение (*image*) – снимок активного окна или объекта до и после выполнения определенного действия программой тестирования;

– файл (*file*) – файл с дополнительными данными, например, генерируемыми в процессе работы программы тестирования;

– ссылка (*link*) – ссылка на файл или другой ресурс.

В случае наличия ошибок или несоответствия описанному поведению СЗИ при анализе журнала работы программы тестирования на основе выводимых событий и соответствующих им дополнительных данных определяется последовательность действий, которая привела к возникновению данных ошибок.

Рекомендации по реализации проверок конкретных функций программно-аппаратного СЗИ зависят от вида рассматриваемого средства защиты при разделении по реализуемым функциям безопасности. Поэтому выработаем рекомендации для одного из видов – средств криптографической защиты информации. Для других видов средств защиты рекомендации будут отличатся в части перечня тестируемых функций.

Программы тестирования для рассматриваемых средств защиты должны состоять из проверок корректности следующих функций безопасности и нецелевых функций:

– генерация симметричных ключей (для различных алгоритмов и длин ключей);

– генерация ключевых пар (для различных алгоритмов и длин ключей);

– экспорт и импорт симметричных ключей (для всех поддерживаемых алгоритмов и длин ключей);

– экспорт и импорт открытых ключей ключевых пар (для всех поддерживаемых алгоритмов и длин ключей);

– зашифрование и расшифрование файлов (всеми поддерживаемыми алгоритмами);

– генерация и проверка ЭП (всеми поддерживаемыми алгоритмами);

– удаление симметричных ключей (для всех поддерживаемых алгоритмов и длин ключей);

– удаление ключевых пар (для всех поддерживаемых алгоритмов и длин ключей);

– удаление только открытых ключей ключевых пар (для всех поддерживаемых алгоритмов и длин ключей);

– удаление только закрытых ключей ключевых пар (для всех поддерживаемых алгоритмов и длин ключей).

Для тестирования описанных функций СЗИ необходимо реализовать и использовать функции проверок, приведенные в Приложении А. Данные проверки необходимо выполнить сначала с постоянно подключенной аппаратной компонентой СЗИ в соответствующий интерфейс СВТ. Затем выполнять ее переподключение в процессе работы и после выполнения каждой из этих проверок с помощью коммутатора канала интерфейса подключения аппаратной компоненты СЗИ к СВТ, предложенного в разделе 3.2. Например, для устройств с USB-интерфейсом подключения можно применять разработанный коммутатор USB-канала. Это связано с тем, что при эксплуатации мобильных программно-аппаратных СЗИ реальным пользователем в ИС извлечение и подключение аппаратной компоненты может происходить в любой момент времени, таким образом, проведение проверок с подключением и отключением этой компоненты является обязательным. Также в некоторых ситуациях применение коммутатора может быть необходимо из-за особенностей аппаратной компоненты СЗИ.

Для использования коммутатора в программах тестирования должен выполняться вызов утилиты командной строки, например, SwitcherConsole.exe (см. раздел 3.2) с передаваемыми ей параметрами:

– для подключения аппаратной компоненты *Sys.OleObject("WScript.Shell").Run('SwitcherConsole.exe on "Switcher. 0“)*, где, «Switcher. 0» – имя используемого коммутатора;

– для отключения аппаратной компоненты – *Sys.OleObject("WScript.Shell").Run('SwitcherConsole.exe off "Switcher. 0“)*.

Программа тестирования должна запускаться со следующими параметрами: *<TESTCOMPLETE_PATH> <PROJECT_PATH> /r /p:Project /u:KeysEncSign /rt:KeysEncSign /e*, где KeysEncSign – название программы тестирования; TESTCOMPLETE_PATH – полный путь до каталога установки TestComplete или TestExecute (средства запуска программ тестирования TestComplete), например: «C:\Program Files (x86)\SmartBear\TestComplete 10\Bin\TestComplete.exe»; PROJECT_PATH – путь до каталога с проектом Project, например: «D:\TestCompleteProjects\Project\Project.pjs»; /r – автоматический запуск программ тестирования при открытии проекта; /p – указание имени проекта (Project); /u – указание имени программы тестирования (KeysEncSign); /rt – указание имени для запуска конкретной функции при откры-

тии программы тестирования (KeysEncSign); /e – автоматическое закрытие среды автоматизации после окончания работы программы тестирования.

Результатом выполнения программы тестирования должен являться отчет в формате htm, в котором последовательно описаны проведенные действия и результаты их выполнения. При возникновении ошибки должна быть возможность изучить действия, которые производились программой до этого момента, путем анализа снимков экрана и сопровождающей их дополнительной информацией.

При возникновении ошибок в работе одной из функций безопасности программа тестирования должна либо завершать свое выполнение, либо продолжать дальнейшее выполнение после запуска функции отслеживания и обработки возникновения ошибок – ErrorEventAndRecovery. В ходе работы программ тестирования программно-аппаратное средство защиты может находиться в различных состояниях (см. разделы 2.1-2.3). Поэтому для корректного продолжения работы программы тестирования после завершения каждой проверки, если это необходимо, осуществляется либо переход СЗИ в другое состояние, требуемое для выполнения следующей проверки, либо возврат в состояние, в котором это средство защиты находилось до начала выполнения текущей проверки (см. раздел 3.3). Для этого рекомендуется использовать функцию ErrorEventAndRecovery, которая может устанавливать глобальную переменную (needRecovery = true), а при вызове любой функции программы тестирования будет проверяться значение этой переменной. При установленной переменной (needRecovery == true) перед выполнением этой функции нужно вызвать функцию возврата аппаратной компоненты в начальное/эталонное состояние recoveryWork(), например, выполнить удаление всех ключей или форматирование устройства. После возврата к начальному состоянию глобальную переменную необходимо сбросить (needRecovery = false).

При разработке программ тестирования необходимо учитывать, что СЗИ даже одного вида, но различных производителей могут предоставлять неспецифичные функции, индивидуальные для каждого такого средства. При этом разработка и использование программ тестирования для таких функций выполняется аналогично рассмотренному выше примеру, за исключением некоторых особенностей. Так, в случае, если существует необходимость тестирования одной из функций безопасности (например, регистрации пользователя в СКЗИ) на нескольких средствах вычислительной техники, то запущенная в среде одного СВТ программа тестирования не сможет продолжить свое выполнение в среде другого. Кроме того, аппаратная компонента должна быть переподключена к этому другому СВТ. Для проведения автоматического тестирования такой проверки можно либо адаптировать используемый коммутатор канала интерфейса подключения аппаратной компоненты СЗИ и подключать его одновременно к нескольким СВТ, тем самым управляя подключением устройства между средствами вычислительной техники. Либо необходимо использовать несколько ВМ, применяя средство виртуализации, запущенное в среде ОС одного СВТ, и предложенный алгоритм тестирования из раздела 3.2. В обоих случаях необходимо использовать несколько программ тестирования: программы, функционирующие на каждом СВТ или ВМ, а также для второго случая – программу, работающую в среде ОС СВТ и выполняющую переключение аппаратной компоненты СЗИ между ВМ в среде виртуализации [28].

Также необходимо учитывать особенности тестирования функции безопасности, которые могут быть реализованы в СЗИ до запуска пользовательской сессии ОС СВТ (например, для СКЗИ – это может быть функция защищенного локального входа в ОС). Для тестирования таких функции безопасности автоматическим способом также можно использовать средства виртуализации и предложенный алгоритм тестирования функций безопасности программно-аппаратных СЗИ с использованием средств виртуализации (см. раздел 3.2). При этом необходимо применять несколько программ тестирования: программа, функционирующая в ОС СВТ и выполняющая тестируемую функцию безопасности с различными параметрами на ВМ (например, производящая идентификацию и аутентификацию пользователя) и программа, автоматически запускаемая в начале пользовательской сессии и сообщающая первой программе об ее успешном запуске. При использовании только одной программы, функционирующей в ОС СВТ, нельзя определить корректность завершения выполняемой функции безопасности в ВМ (например, идентификации и аутентификации). Это связано с тем, что программа тестирования может работать только с нативными объектами ОС СВТ и не может перехватывать и анализировать результаты своих действий в ОС ВМ, в том числе при использовании таких технологий, как VMware Unity mode для средств виртуализации VMware [113], Seamless mode в VirtualBox [110], Coherence для средств виртуализации Parallels [97].

Последняя рекомендация, без учета необходимости передачи результатов работы функции безопасности в ОС, подходит также для мобильных программно-аппаратных СЗИ, функционирующих независимо от ОС СВТ. Помимо этого для таких СЗИ также применимы рекомендации из следующего раздела, предложенные для стационарных СЗИ, функционирующих также независимо от ОС СВТ.

На основании выше описанного можно привести рекомендации по практической реализации средств тестирования функций безопасности мобильных программно-аппаратных средств защиты информации, взаимодействующих со средой ОС СВТ. Средство тестирования таких СЗИ рекомендуется:

1. Реализовывать с помощью средства автоматизации TestComplete, наиболее подходящего для автоматического тестирования функций безопасности данного вида.
2. Разрабатывать с учетом необходимости следующего:
   – Одновременного тестирования функций безопасности и нецелевых функций СЗИ.
   – Проведения тестирования сначала с постоянно подключенной аппаратной компонентой, а затем ее переодключением в процессе работы и после выполнения каждой проверки.
   – Применения соответствующего коммутатора канала интерфейса подключения аппаратной компоненты СЗИ к СВТ.
   – Использования функции отслеживания и обработки ошибок, позволяющей в случае необходимости после завершения проверки перевести СЗИ в другое состояние, требуемое для выполнения следующей проверки.
   – Использования нескольких программ для проведения тестирования одной функции безопасности на нескольких СВТ: либо подключая одновременно к нескольким СВТ адаптированный коммутатор канала, либо используя несколько ВМ через запущенное в среде ОС одного

из СВТ средство виртуализации. Таким образом, необходимо использование программ, функционирующих на каждом СВТ или ВМ, а также для второго случая – программы, работающей в среде ОС СВТ и выполняющей переключение аппаратной компоненты СЗИ между ВМ в среде виртуализации.

– Использования нескольких программ при тестировании функций безопасности, реализованных в СЗИ до запуска пользовательской сессии ОС СВТ: программы, функционирующей в ОС СВТ и выполняющей тестируемую функцию безопасности с различными параметрами на ВМ и программы, автоматически запускаемой в начале пользовательской сессии и сообщающей первой программе об ее успешном запуске.

– Предоставления результатов выполнения программ тестирования в виде отчета в формате htm с последовательным описанием проведенных действий и результатов их выполнения, включая снимки экрана при тестировании.

На основании данных рекомендаций выполнена разработка программы и методики испытаний, а также программ тестирования для мобильных программно-аппаратных СКЗИ, включающих универсальные проверки и проверки, специфичные для ПСКЗИ ШИПКА (см. раздел 1.2) – средства криптографической защиты информации, функции безопасности которого реализованы на базе мобильной аппаратной компоненты с USB-интерфейсом подключения в ОС СВТ. Данные программы тестирования соответствуют требованиям к средствам тестирования из раздела 3.3 и используют коммутатор USB-канала (предложенный в разделе 3.4) для автоматического подключения и отключения СКЗИ к/от СВТ [31, 32].

Фрагменты листинга программы тестирования утилиты «Ключи. Шифрование и подпись файлов», входящей в состав ПСКЗИ ШИПКА (TestedApps: ACshEncSig, тип программы тестирования: Script, язык: *JScript*, проект: Shipka, программа тестирования KeysEncSign), а также листингов функций обработки ошибок при работе программы тестирования и возврата ПСКЗИ ШИПКА к эталонному состоянию приведены в Приложении В, листинги В.1 и В.2 соответственно. А примеры результата работы программы тестирования KeyEncSign и программы тестирования, осуществляющей идентификацию и аутентификацию пользователя с использованием ПСКЗИ ШИПКА в ВМ приведены на Рисунках 3.6 и 3.7 соответственно.

Рекомендации по практической реализации средств тестирования функций безопасности стационарных программно-аппаратных средств защиты информации

Программно-аппаратное СЗИ со стационарной аппаратной компонентой, функционирующее в ОС СВТ, представляет собой комплексное решение, состоящее из аппаратной компоненты и взаимодействующего с ней из ОС специального программного обеспечения. При этом в средстве защиты могут быть реализованы функции безопасности, для корректного завершения которых требуется выполнение каких-либо действий, например, в средствах разграничения доступа для выполнения процедуры идентификации необходимо предъявлять аппаратные идентификаторы. В соответствии с этим, а также на основе требований к средствам тестирования из раздела 3.3 для автоматического тестирования рассматриваемых СЗИ требуется:

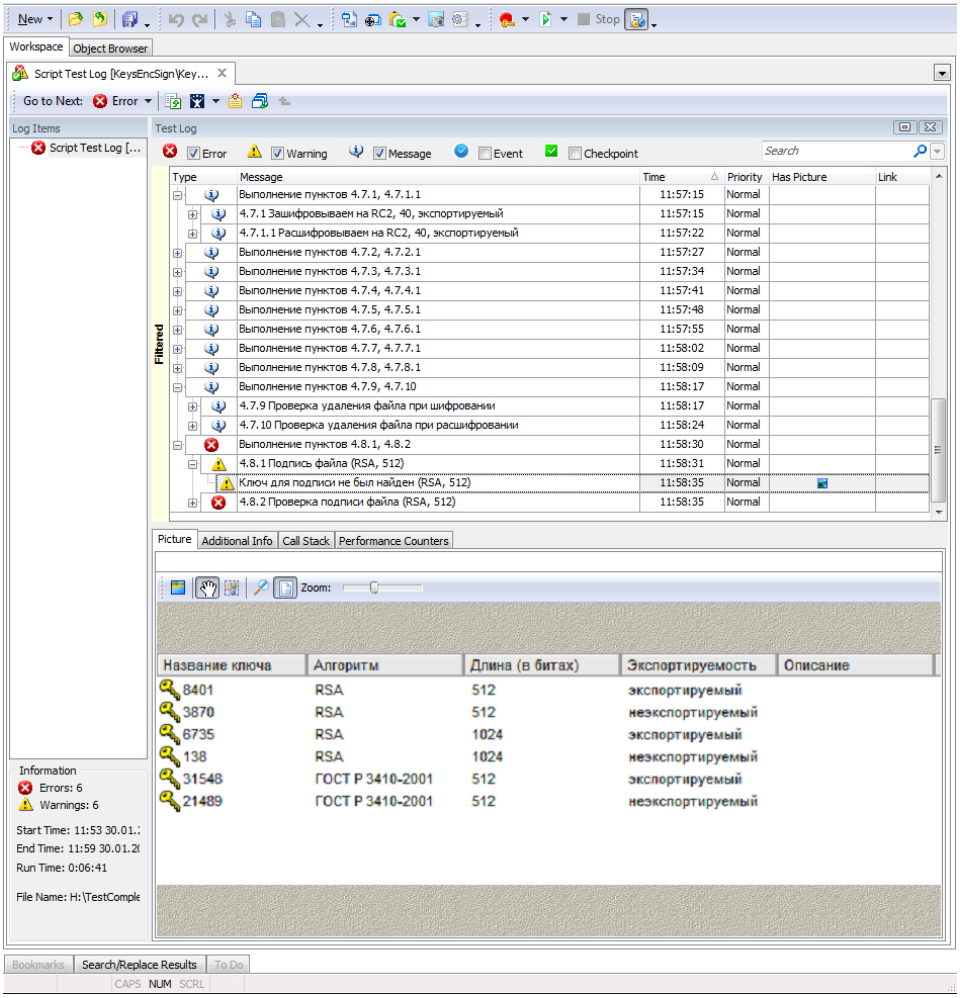


Рисунок 3.6 – Результат работы программы тестирования ПСКЗИ ШИПКА

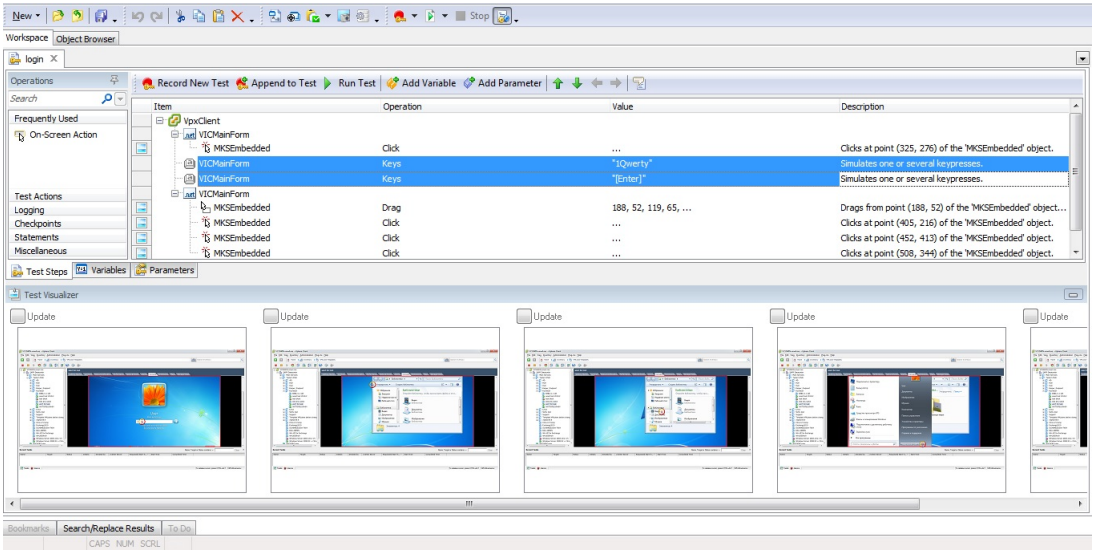


Рисунок 3.7 – Программа тестирования, выполняющая идентификацию и аутентификацию пользователя с использованием ПСКЗИ ШИПКА в ВМ

– осуществлять перенос стационарной аппаратной компоненты из одного СВТ в другое;

– встроить средства тестирования как в ОС СВТ (для функций безопасности, выполняющихся в ОС), так и в ОС СЗИ аппаратной компоненты (для функций, выполняющихся независимо от ОС), обеспечивать возможность фиксации результатов их применения;

– разработать удовлетворяющие требованиям раздела 3.3 программы тестирования.

При этом, как было отмечено в выводах раздела 3.3, тестирование функций безопасности, реализованных на базе стационарной аппаратной компоненты, на физических СВТ затруднено и требует применения вспомогательных программно-аппаратных средств или специализированных автоматизированных комплексов, чья сложность и стоимость может значительно превышать расходы на разработку самих средств защиты. Кроме того, одной из целей тестирования рассматриваемых СЗИ является проверка корректности функций безопасности при большой нагрузке (например, для средств разграничения доступа – при большом количестве запросов на доступ) и на большом количестве поддерживаемых программных платформ – дистрибутивах Linux различной разрядности и архитектуры), например, AltLinux 6.0 СПТ и 7.0.5 x86/x86_64; Astra Linux SE 1.3 x86_64; Debian Linux 7.6.0 x86_64; RedHat Enterprise Linux 5.1, 5.4, 5.7, 6.1 x86_64, 6.4 x86_64 и 7.0 x86_64; Rosa Linux 1.0 Cobalt x86/x86_64; Ubuntu Linux 12.04.5 x86/x86_64.

При тестировании функции безопасности, взаимодействующих с ОС СВТ, особых требований к «чистоте» эксперимента нет. В связи с этим, а также на основе исследований из Главы 2, для тестирования таких функций безопасности необходимо применить предложенный в разделе 3.2 алгоритм тестирования программно-аппаратных СЗИ с использованием средств виртуализации. При этом само тестирование требуется проводить на специализированном СВТ, все компоненты которого поддерживают технологию Intel VT-d или AMD IOMMU. В качестве средства виртуализации можно использовать одно из средств, в котором программная реализация используемых в тестируемом СЗИ интерфейсов подключения аппаратных устройств, а также используемых компонент ВМ, аналогичных компонентам физического СВТ (BIOS, «аппаратные» прерывания и другие), соответствуют существующим стандартам и спецификациям. Помимо этого необходимо, чтобы в средстве виртуализации была реализована возможность перенаправления в ВМ не только самой стационарной аппаратной компоненты, но и всех дополнительных аппаратных компонент, которые являются мобильными, например, аппаратных идентификаторов. Всем описанным условиям соответствует средство виртуализации KVM (с управлением через libvirt), поэтому рекомендуется использовать именно его.

Непосредственно перед первым запуском каждой ВМ в нее необходимо перенаправить стационарную аппаратную компоненту, руководствуясь разделом 3.2. Например, для PCI-интерфейса подключения аппаратной компоненты необходимо выполнить ее привязку по известным идентификационным номерам производителя и устройства (англ. Vendor ID и Product ID, VID и PID) к драйверу pci_stub или vfio-pci (см. Приложение Б).

В последствии для автоматического перенаправления аппаратной компоненты СЗИ необходимо добавить ее в xml-файл с описанием запускаемой ВМ.

В процессе тестирования для проведения проверок некоторых функций безопасности в ВМ также необходимо перенаправлять дополнительные мобильные аппаратные компоненты, описываемые по известным идентификационным номерам производителя и устройства (VID и PID)

или по расположению на соответствующей используемому интерфейсу шине в отдельном xml-файле описания *dst-hardware-interface-passthrough.xml* (см. Приложение Б).

Для переподключения дополнительных мобильных аппаратных компонент СЗИ в процессе тестирования вместо коммутатора соответствующего интерфейса необходимо использовать стандартные возможности среды виртуализации (см. раздел 3.2):

– *«virsh attach-device VM_NAME dst-hardware-interface-passthrough.xml»* – для подключения устройств из файла описания *dst-hardware-interface-passthrough.xml* к ВМ с именем VM_NAME;

– *«virsh detach-device VM_NAME dst-hardware-interface-passthrough.xml»* – для отключения устройств из файла описания *dst-hardware-interface-passthrough.xml* от ВМ с именем VM_NAME.

Разрабатываемые программы тестирования функций безопасности программно-аппаратных СЗИ, реализованных на базе стационарной аппаратной компоненты, должны проводить проверки в два этапа: вначале до загрузки ОС ВМ в отношении функций безопасности, выполняющихся независимо от ОС СВТ, а затем непосредственно в ОС ВМ в отношении функций безопасности, выполняющихся в ОС СВТ, в обоих случаях с учетом возможности подключения различных дополнительных аппаратных компонент.

На первом этапе тестирование должно проводиться до загрузки ОС ВМ, в соответствии с чем в случае использования физических СВТ с большой вероятностью пришлось бы вносить изменения как в средства тестирования, так и в программную, а также аппаратную компоненты самого программно-аппаратного СЗИ (например, расширять доступную память, встраивать и адаптировать средства автоматизации). Поэтому рекомендуется применение средств виртуализации, которое позволяет частично избежать этого: основная часть средств и программ тестирования будут функционировать в ОС со средой виртуализации и в ОС ВМ, в которых без ограничений можно также сохранять результаты работы автоматических проверок, а в ОС СЗИ будет внедряться лишь малая часть программ тестирования, которые должны взаимодействовать с программами из ОС СВТ. Если в качестве ОС СЗИ используется полноценная среда Linux со встроенными интерпретаторами, позволяющими писать программы тестирования (bash, python), становится возможным исключить необходимость адаптации встраиваемых в ОС СЗИ средств автоматизации, реализовав необходимые проверки с использованием встроенных возможностей Linux. При этом на первом этапе вследствие функционирования программ тестирования в ОС со средой виртуализации и в ОС СЗИ становятся вычислимыми не только функции безопасности, функционирующие в ОС СЗИ, но и функционирующие до нее сразу после работы BIOS ВМ (SeaBIOS).

На втором этапе тестирования проверки проводятся в отношении функций безопасности, взаимодействующих с ОС СВТ, для чего необходимо использовать программы тестирования в ОС ВМ и в ОС со средой виртуализации, последняя из которых используется в качестве средства управления и консолидации результатов проверок.

Программы тестирования должны содержать возможность автоматической сборки программной компоненты СЗИ для различных поддерживаемых программных платформ перед этапами ее автоматической установки в ОС и запуска проверок функций безопасности. В связи

с этим будет возможно организовать регрессионное тестирование, при котором любые изменения в исходных программных кодах СЗИ автоматически проверяются на предмет нарушения корректно работающих ранее функций безопасности.

Необходимо также отметить, что разработанные с учетом рекомендаций выше программы тестирования из-за отсутствия какой-либо привязки к средствам виртуализации могут без значительных изменений быть применимы и для физических СВТ в том числе, если предъявляются требования к «чистоте» эксперимента. При этом программа тестирования, которая запускалась в ОС со средой виртуализации, должна выполняться на некотором СВТ, с которого осуществляется управление процессом тестирования (по сети) и консолидация результатов, программы тестирования в ОС ВМ – в ОС тестируемых физических СВТ, а программа тестирования в ОС СЗИ так и будет выполняться в этой среде. В данном случае вместо автоматического будет возможно провести только автоматизированное тестирование даже в случае использования предложенного в разделе 3.4 коммутатора для подключения/отключения дополнительных аппаратных компонент вследствие того, что:

– не существует возможности провести автоматические проверки функций безопасности, выполняющихся до ОС СВТ и СЗИ (например, перехвата загрузки ОС СВТ);

– не существует возможности провести автоматические проверки функций безопасности выполняющихся до запуска пользовательской сессии в ОС СВТ (например, для подсистемы разграничения доступа – идентификации и аутентификации в ОС при входе в графический пользовательский интерфейс);

– для фиксации результатов всех автоматических проверок в стационарной аппаратной компоненте может быть недостаточно памяти, а при возможной перезагрузке в ходе тестирования могут не сохраниться последние результаты;

– отсутствуют вспомогательные средства тестирования, с помощью которых возможно автоматически осуществлять встраивание стационарной аппаратной компоненты в СВТ.

На основании выше описанного можно привести рекомендации по практической реализации средств тестирования функций безопасности стационарных программно-аппаратных средств защиты информации, взаимодействующих со средой ОС СВТ. Средство тестирования таких СЗИ должно учитывать необходимость:

– Использования специализированных СВТ с поддержкой технологии Intel VT-d или AMD IOMMU.

– Применения средства виртуализации KVM, так как в нем программная реализация используемых в тестируемом СЗИ интерфейсов подключения аппаратных устройств, а также используемых компонент ВМ, аналогичных компонентам физического СВТ (BIOS, «аппаратные» прерывания и другие), соответствуют существующим стандартам и спецификациям. А также в данном средстве виртуализации реализована возможность перенаправления в ВМ не только самой стационарной аппаратной компоненты, но и всех дополнительных аппаратных компонент, которые являются мобильными, например, аппаратных идентификаторов.

Средство тестирования рассматриваемых СЗИ должно реализовывать следующие возможности:

– Перед первым запуском каждой ВМ перенаправлять в нее стационарную аппаратную компоненту путем привязки по VID и PID к соответствующему драйверу.

– При необходимости перенаправлять дополнительные мобильные аппаратные компоненты путем прописывания их в соответствующем xml-файле. При этом использовать стандартные возможности среды виртуализации для переподключения дополнительных мобильных аппаратных компонент.

– Проводить проверки в два этапа: вначале до загрузки ОС ВМ в отношении функций безопасности, выполняющихся независимо от ОС СВТ, а затем непосредственно в ОС ВМ в отношении функций безопасности, выполняющихся в ОС СВТ, в обоих случаях с учетом возможности подключения различных дополнительных аппаратных компонент.

– Выполнять автоматическую сборку программной компоненты СЗИ для различных поддерживаемых программных платформ перед этапами ее автоматической установки в ОС и запуска проверок функций безопасности в целях организации регрессионного тестирования, при котором любые изменения в исходных программных кодах СЗИ автоматически проверяются на предмет нарушения корректно работающих ранее функций безопасности.

Дальнейшие рекомендации по проверкам рассматриваемых средств защиты зависят от их вида по реализуемым функциям безопасности, поэтому приведем рекомендации для одного из видов – средств разграничения доступа в ОС, состоящих из модуля доверенной загрузки, функционирующего до старта сессии пользователя, и ПО разграничения доступа в ОС. Для СЗИ других видов также применимы все перечисленные рекомендации, и только третья зависит от набора реализуемых средством защиты функций безопасности и будет являться специфичной для каждого вида СЗИ.

Программы тестирования для таких средств защиты в ОС ВМ должны выполнять следующие действия:

1. Автоматически устанавливать и настраивать ПО разграничения доступа в зависимости от ОС ВМ, в которой проводится тестирование: установка пакетов, настройка компонент ОС и загрузчика, базовая настройка СЗИ.

2. Перезагружать ВМ и проверять работоспособность СЗИ с базовыми настройками (например, корректность активизации СЗИ, идентификации и аутентификации при входе в ОС), в ходе которой программа тестирования в ОС со средой виртуализации перенаправляет и предъявляет дополнительные аппаратные компоненты, (например, аппаратные идентификаторы в ВМ).

3. Генерировать объекты файловой системы и правила для проверки функций безопасности СЗИ с различным параметрами:

– для всесторонней проверки идентификации и аутентификации с различными типами аппаратных идентификаторов и возможностью удаленного входа в ОС;

– с учетом всевозможных сочетаний атрибутов доступа, а также правил рекурсии с которыми они используются для файлов и директорий в дискреционной политике управления доступом;

– для различных уровней доступа пользователей и конфиденциальности объектов в мандатной политике управления доступом;

– с учетом различных способов осуществления доступа субъектов к объектам или изменения целостности последних при работе функции контроля целостности;

– все перечисленное в сочетании охватывает все возможные состояния программной и аппаратной компонент СЗИ и возможные функции переходов (возврат в начальное состояние при очистке списков прав доступа, контроля целостности и пользователей; форматирование аппаратной компоненты; переход в «мягкий» режим для пополнения списков пользователей, прав доступа и контроля целостности и так далее).

4. Запускать проверки функций безопасности (порядка 100000 запросов на доступ при работе дискреционного и мандатного управления доступом, а также динамического контроля целостности с различными настройками и сочетанием функций защиты) и фиксировать возможные несоответствия.

5. Отправлять результаты проверок программе тестирования в ОС среды виртуализации (для каждой проверки однозначно определяется результат: *«[Test Passed]»* – проверка пройдена успешно; *«[Test Failed]»* – возникла ошибка в функции безопасности; *«[Test Error]»* – при проведении проверки возникла ошибка в программе тестирования, которую необходимо исправить и запустить проверку заново).

Рекомендации, сформулированные для аппаратной компоненты рассматриваемого вида программно-аппаратных СЗИ, также подходят для стационарных СЗИ, функционирующих независимо от ОС СВТ.

На основании приведенных рекомендаций с использованием предложенного в разделе 2.5 способа разработаны программы тестирования для подсистемы разграничения доступа ПАК СЗИ НСД «Аккорд-X», построенной на базе аппаратного модуля доверенной загрузки «Аккорд-АМДЗ». Данные программы тестирования позволяют осуществить проверку всех функций безопасности при различных сочетаниях и во всевозможных состояниях, в которых потенциально эти функции могут быть вычислимы. Запуск и результаты работы программ тестирования ПАК СЗИ НСД «Аккорд-X» на одной из ВМ приведены в Листинге В.3 из Приложения В. После проведения такого тестирования достаточно проверить возможность функционирования используемого в комплексе АМДЗ на физических СВТ в части возможного несоответствия некоторых его составляющих и параметров (BIOS, питания PCI-устройств и других) существующим спецификациям и стандартам (см. раздел 3.1). Большую часть разработанных программ тестирования, за исключения специфичных мест с настройкой конкретных программно-аппаратных СЗИ, можно потенциально использовать для широкого круга аналогичных по выполняемым функциям безопасности средств защиты – аппаратных модулей доверенной загрузки «Соболь», ПАК СЗИ НСД «Аккорд-Win32» и «Аккорд-Win64», «Secret Net» и «Secret Net LSP» или других.

Рекомендации по практической реализации средств верификации программно-аппаратных средств защиты информации

Разработанные выше программы позволяют автоматически тестировать функции безопасности программно-аппаратных СЗИ. Однако результаты работы этих программ все равно приходится анализировать вручную для принятия решения о необходимости устранения выявленных

ошибок в функциях безопасности СЗИ или об успешном завершении процесса верификации. При этом в разделе 2.4 для принятия решения по результатам тестирования был предложен алгоритм верификации программно-аппаратных СЗИ, реализующих функции безопасности, основанный на классификации обнаруженных ошибок, анализе степени их критичности и влияния на защищенность системы или данных. Однако применение данного алгоритма предполагает достаточно большой объем ручной работы. В соответствии с этим для сокращения временных затрат на процесс верификации необходимо разработать программную реализацию упомянутого алгоритма, предоставляющую возможность автоматического решения задачи оценки критичности выявленных в ходе тестирования ошибок в функциях безопасности, предварительно сформулировав рекомендации к ее практической реализации.

Входными данными программы верификации должны являться результаты тестирования функций безопасности программно-аппаратных СЗИ в определенном заранее формате. При поступлении входных данных программа верификации должна позволять оценить критичность выявленных в ходе тестирования ошибок и степень их влияния на защищенность ИС. При этом входные данные должно быть можно задавать либо вручную (определяя критичность всех выявленных ошибок), либо передавать на вход результаты разработанных в соответствии с рекомендациями из предыдущих разделов программ тестирования функций безопасности программно-аппаратных СЗИ. В последнем случае критичность ошибок должна определяться автоматически и выставляться в результатах программ тестирования в зависимости от проверок функций безопасности, которые завершились с отрицательным результатом.

В качестве объектов оценки должны выступать различные версии одного и того же СЗИ, протестированные во всевозможных ОС и с различными исполнениями аппаратной компоненты. В результате этого становится возможным сравнить тестируемое программно-аппаратное СЗИ как с предыдущими его версиями для проведении регрессионного тестирования, так и с некоторой абстрактной «эталонной» версией, в которой не выявлено никаких ошибок, для оценки степени влияния найденных ошибок на качество средства защиты.

При этом необходимо предусмотреть два режима работы программы верификации [33, 105]: базовый – оценка СЗИ проводится только на основе выявленных ошибок разного уровня критичности; расширенный – оценка СЗИ проводится также на основе критичности ошибок, но с учетом уровня критичности функций безопасности, в которых они были выявлены.

Базовый режим работы должно быть возможно использовать для любых программно-аппаратных СЗИ, тогда как расширенный – в отношении тех, для которых разработаны программы тестирования в соответствии с рекомендациями в предыдущих разделах. Оба режима работы должны позволять ранжировать объекты оценки по степени влияния выявленных ошибок на качество программно-аппаратного СЗИ, а также, кроме оценки непосредственно критичности выявленных ошибок, учитывать и их количество. Применение того или иного способа оценки (только по критичности ошибок или по критичности и количеству) зависит от конкретных целей верификации.

В программе верификации должен применяться существующий математический аппарат теории оптимизации и принятия решения, предложенный в разделе 2.4. При этом будет ре-

шаться задача оценки критичности выявленных ошибок в функциях безопасности программно-аппаратного СЗИ и ранжирования альтернатив (версий СЗИ) по критериям оценки. Иерархическая структура данной задачи приведена на Рисунке 3.8 и состоит из цели, частных критериев (наличия ошибок трех уровней критичности) и оцениваемых альтернатив [105].

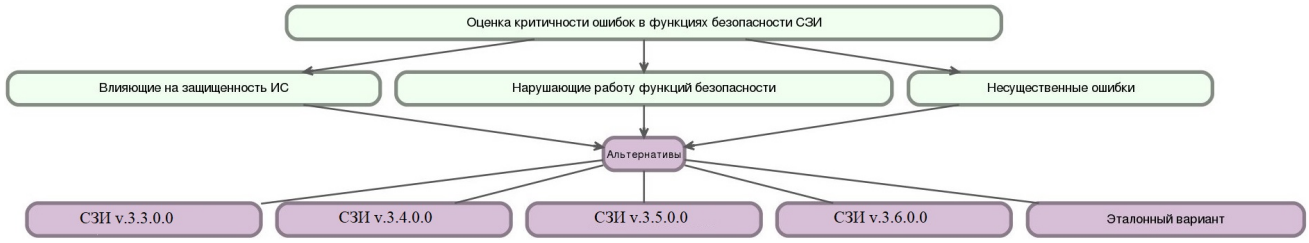


Рисунок 3.8 – Визуализация иерархической структуры решаемой задачи оценки критичности ошибок СЗИ: цель, частные критерии и альтернативы

В базовом режиме работы программа верификации должна учитывать только критичность выявленных в ходе тестирования ошибок в функциях безопасности СЗИ и их количество. При этом для работы этой программы необходимо задать приведенные выше количественные данные о выявленных ошибках (результаты работы программы тестирования), а также определить шкалу критичности, описывающую отношения «важности» (превосходства) одного критерия над другим. В соответствии со шкалой Саати [94] в качестве результата попарного сравнения превосходства ошибок принятых уровней критичности целесообразно задать следующие значения: критические ошибки в сравнении с ошибками в функциях безопасности – 3 (среднее превосходство), критические ошибки в сравнении с незначительными – 9 (абсолютное превосходство), ошибки в функциях безопасности в сравнении с некритическими – 7 (сильное превосходство). По заданным значениям оценки превосходства программа верификации должна автоматически вычислить числовые значения весовых коэффициентов для критериев оценки. Важно отметить, что шкала критичности должна быть определена только один раз и в дальнейшем может без изменений применяться для решения новых аналогичных задач оценки критичности ошибок программно-аппаратных СЗИ [33, 105].

На основе входных данных программа верификации должна позволять автоматически оценивать критичность выявленных в ходе тестирования ошибок в альтернативах. При использовании предложенного в разделе 2.4 математического аппарата также учитывается количественные показатели выявленных ошибок.

В расширенном режиме работы программа верификации должна позволять учитывать не только критичность ошибок и их количество, но также и критичность функций безопасности, в которых эти ошибки были выявленных в ходе тестирования функций безопасности программно-аппаратного СЗИ. В данном режиме в иерархическую структуру решаемой задачи добавляется новый уровень частных критериев, характеризующих тестируемые функции безопасности СЗИ, для которого критерии с ошибками будут являться уже вложенными локальными критериями (для каждого критерия вышестоящего уровня иерархии). В таких условиях например, для СКЗИ, целесообразно рассматривать следующие функции безопасности, которые будут представлять критерии вышестоящего уровня: идентификация и аутентификация для доступа к функциям

безопасности; генерация ключей и ключевых пар; зашифрование и расшифрование; генерация и проверка электронной подписи; экспорт и импорт открытых ключей; экспорт и импорт закрытых или симметричных ключей [33, 105].

На основании выше описанного можно привести рекомендации по практической реализации средства верификации программно-аппаратных СЗИ [105]:

– Входными данными программы верификации должны являться результаты тестирования функций безопасности программно-аппаратных СЗИ в определенном заранее формате.

– Программа должна позволять оценить критичность выявленных в ходе тестирования ошибок и степень их влияния на защищенность ИС.

– В качестве объектов оценки должны выступать различные версии одного и того же СЗИ, протестированные во всевозможных ОС и с различными исполнениями аппаратной компоненты. В результате этого становится возможным сравнить тестируемое программно-аппаратное СЗИ как с предыдущими его версиями для проведении регрессионного тестирования, так и с некоторой абстрактной «эталонной» версией, в которой не выявлено никаких ошибок, для оценки степени влияния найденных ошибок на качество средства защиты.

– Должно быть предусмотрено два режима работы программы верификации базовый и расширенный, когда оценка СЗИ проводится только на основе выявленных ошибок разного уровня критичности или на основе критичности ошибок, но с учетом уровня критичности функций безопасности, в которых они были выявлены, соответственно.

На основании предложенных рекомендаций разработана программа «Верификатор программно-аппаратных СЗИ», реализующая рассмотренные выше базовый и расширенный режимы работы. Таким образом, в разделе приведены рекомендации по практической реализации программы верификации для функций безопасности программно-аппаратных СЗИ. На основании данных рекомендаций с использованием предложенного в разделе 2.4 алгоритма верификации программно-аппаратных СЗИ разработана программа «Верификатор программно-аппаратных СЗИ», которая представляет собой систему поддержки принятия решений [96], позволяющую автоматически оценить критичность выявленных ошибок в функциях безопасности программно-аппаратных СЗИ после использования программ тестирования. При этом такой программе «Верификатор программно-аппаратных СЗИ» могут анализироваться как результаты тестирования различных версий СЗИ между собой в рамках проведения регрессионного тестирования, так и в сравнении с абстрактной «эталонной» версией для оценки степени влияния выявленных ошибок на качество средства защиты той или иной версии (или при тестировании в тех или иных условиях). Кроме того, для учета различных целей верификации в разработанной программной реализации предусмотрено два режима работы, результаты которых могут отличаться в зависимости от «важности» функций безопасности, в которых выявлены ошибки разного уровня критичности. Вследствие зависимости расширенного режима работы от программно-аппаратного СЗИ в части реализуемых функций безопасности и их значимости в рамках верификации, его возможно применять в отношении средств защиты, для которых ранее в разделе разработаны программы тестирования (СКЗИ и подсистемы разграничения доступа

различных производителей). Базовый режим работы можно без изменений применять в отношении любых программно-аппаратных, а также программных СЗИ.

3.6. Выводы

В главе получены следующие результаты:

1. Показано, что для корректной работы всех видов функций безопасности программно-аппаратных СЗИ необходимо проверять совместимость их аппаратной и программной компоненты с аппаратной платформой. Для этого требуется проводить тестирование средств защиты на множестве СВТ с целью выявления возможных особенностей и несоответствия определенным стандартам последних, что может повлиять на вычислимость функций безопасности.

2. Разработан алгоритм тестирования функций безопасности программно-аппаратных СЗИ с применением средств виртуализации, позволяющий обеспечить возможность выполнения некоторых функций безопасности в виртуальных машинах при невозможности их выполнения на реальных аппаратных платформах.

3. Сформулированы требования к средствам тестирования функций безопасности программно-аппаратных СЗИ и среде их применения, при соответствии которым проверка каждой функции безопасности проводится во всех состояниях, в которых она потенциально вычислима. Данные требования могут быть выполнены либо вручную, либо с использованием программ тестирования и применения дополнительных средств, реализующих необходимые функции переходов: подключение/отключение к/от СВТ аппаратной компоненты СЗИ и ее перенос в другое СВТ.

4. Сформулированы рекомендации по практической реализации вспомогательных средств для автоматизации тестирования функций безопасности программно-аппаратных СЗИ. На основании предложенных рекомендаций выполнена разработка коммутатора USB-канала, который полностью эмулирует физическое отключение и подключение СЗИ к СВТ как на уровне питания, так и на уровне канала передачи данных. Данный коммутатор может использоваться в программах тестирования для автоматического отключения и подключения СЗИ с USB-интерфейсом к/от СВТ.

5. Предложены рекомендации по практической реализации программного комплекса тестирования различных видов функций безопасности программно-аппаратных СЗИ, состоящего из программ тестирования и программы верификации.

6. Выполнена разработка программого комплекса «Тестирование функций безопасности программно-аппаратных средств защиты информации» (свидетельство о государственной регистрации программы для ЭВМ №2016616332, см. Приложение Г), реализующего предложенные алгоритмы тестирования и верификации функций безопасности программно-аппаратных СЗИ. Применение данного комплекса совместно с коммутатором канала интерфейса подключения аппаратной компоненты СЗИ к СВТ позволяет выполнять автоматическое тестирование различных видов функций безопасности программно-аппаратных СЗИ.

4. Анализ опыта совместного использования разработанного программно-аппаратного комплекса для тестирования СЗИ со сторонними вспомогательными для тестирования средствами и специализированными средствами тестирования СЗИ

В Главах 2-3 предложен способ тестирования функий безопасности программно-аппаратных СЗИ, а также приведено описание реализованых для них средств тестирования (комплекс программ тестирования и вспомогательное средство), в отношении которых в данной главе необходимо:

– исследовать эффект от использования предложенного способа и разработанных средств тестирования по отношению к различным видам функций безопасности программно-аппаратных СЗИ как на количественном, так и на качественном уровне;

– подтвердить практическую возможность использования предложенного способа тестирования к функциям безопасности программно-аппаратных СЗИ;

– оценить возможность применения результатов диссертационного исследования для различных видов функций безопасности и конкретных реализующих их программно-аппаратных средств защиты;

– описать апробацию и реализацию результатов проведенного исследования;

– проанализировать необходимость совместного использования предложенного способа и разработанных средств с существующими сторонними специализированными средствами тестирования в целях упрощения анализа результатов тестирования, а также подтверждения заявленных характеристик аппаратной компоненты СЗИ.

4.1. Методика проведения экспериментальной апробации разработанного способа и основаного на нем программно-аппаратного комплекса для тестирования СЗИ

Рассмотрим практическую возможность использования разработанного в Главе 2 способа тестирования к различным видам функций безопасности нескольких программно-аппаратных СЗИ. На основании полученного результата определим возможность применения ручного, автоматического и автоматизированного способа тестирования к таким СЗИ. В случае невозможности их использования выявим причины, в соответствии с которыми проверки некоторых функций безопасности становятся невыполнимыми, а также необходимость применения вспомогательных средств тестирования для выполнения таких проверок.

В качестве рассматриваемых СЗИ выберем средства, которые реализуют функции безопасности различных видов в соответствии с разделом 1.2, а именно следующие:

– ПСКЗИ ШИПКА – программно-аппаратное СЗИ, функции безопасности которого реализованы на базе мобильной аппаратной компоненты и взаимодействуют со средой ОС СВТ;

– СЗИ НСД «Аккорд-АМДЗ» – программно-аппаратное СЗИ, функции безопасности которого реализованы на базе стационарной аппаратной компоненты и не взаимодействуют с ОС СВТ (выполняются независимо от ОС в составе СВТ).

Первым шагом предложенного в Главе 2 способа тестирования программно-аппаратных СЗИ является определение принципиальной возможности проведения тестирования их функций

безопасности в соответствии с критериями для ручных, автоматических и полуавтоматизированных проверок, а также – частными критериями для различных видов функций безопасности (с мобильной и станионарной аппаратной компонентой).

Обозначим СЗИ НСД «Аккорд-АМДЗ» как $m_{амдз} \in M$. В соответствии с документацией на СЗИ НСД «Аккорд-АМДЗ» [70] – $m_{амдз} \in P_{встр} \subseteq P_{нос}$ и $m_{амдз} \in P_{стац}$. Представим «Аккорд-АМДЗ» в виде конечного детерминированного автомата $\widetilde{m}_{амдз} = (V_{амдз}, I_{амдз}, O_{амдз}, f_{амдз}, g_{амдз})$.

Согласно документации на СЗИ НСД «Аккорд-АМДЗ» [70] $m_{амдз}$ может находиться в следующих состояниях (для упрощения восприятия рассматриваемых состояний далее опустим обозначение *«амдз»* при их описании) – $v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}, v_{13} \in V_{амдз}$:

– контроллер находится в рабочем режиме:

v_0 – не встроен (не установлен в PCI-слот ПК) и не инициализирован (пустая база данных пользователей) – начальное состояние СЗИ;

v_1 – встроен (установлен в PCI-слот ПК) и не инициализирован;

v_2 – встроен и инициализирован (зарегистрированы пользователи с ролями «главный Администратор» и «Пользователь»);

v_3 – не встроен и инициализирован;

v_4 – «главным Администратором» пройдена идентификация/аутентификация в контроллере;

v_5 – «Пользователем» пройдена идентификация/аутентификация в контроллере;

v_6 – «Пользователем» пройден контроль целостности технических и программных средств ПК в контроллере;

v_7 – выполненна доверенная загрузка ОС.

– контроллер находится в технологическом режиме:

v_8 – не встроен и не инициализирован;

v_9 – встроен и не инициализирован;

v_{10} – не инициализирован, выполнена загрузка ОС;

v_{11} – не встроен и инициализирован;

v_{12} – встроен и инициализирован;

v_{13} – инициализирован, выполнена загрузка ОС.

При этом $v_0, v_2, v_4, v_5, v_6, v_7 \in V_{амдз_{фб}}$.

Используя результаты Главы 2 построим граф $G_{m_{амдз}} = (V_{амдз}, E_{амдз})$, соответствующий конечному автомату $\widetilde{m}_{амдз} = (V_{амдз}, I_{амдз}, O_{амдз}, f_{амдз}, g_{амдз})$, на котором множества автомата $I_{амдз}, O_{амдз}, T_{амдз}$ представим в виде дуг, см. Рисунок 4.1.

В соответствии с документацией [70] все переходы из одного состояния в другое с помощью стимулов $I_{амдз}$ могут быть выполнены вручную (некоторые с помощью ручного встраивания и перевода контроллера из технологического режима в рабочий и обратно), то есть $m_{амдз} \in M_p$.

Все переходы $T_{амдз}$, соответствующие дугам графа, кроме (v_0, v_8), (v_8, v_0), (v_3, v_{11}), (v_{11}, v_3), (v_8, v_9), (v_9, v_8), (v_{11}, v_{12}), (v_{12}, v_{11}), (v_0, v_1), (v_1, v_0), (v_3, v_2) и (v_2, v_3) (встраивание/отчуждение контроллера, перевод в технологический/рабочий режим) могут быть выполнены автоматически в том случае, если выполнить внедрение средства автоматизации и программ тестирования в ОС

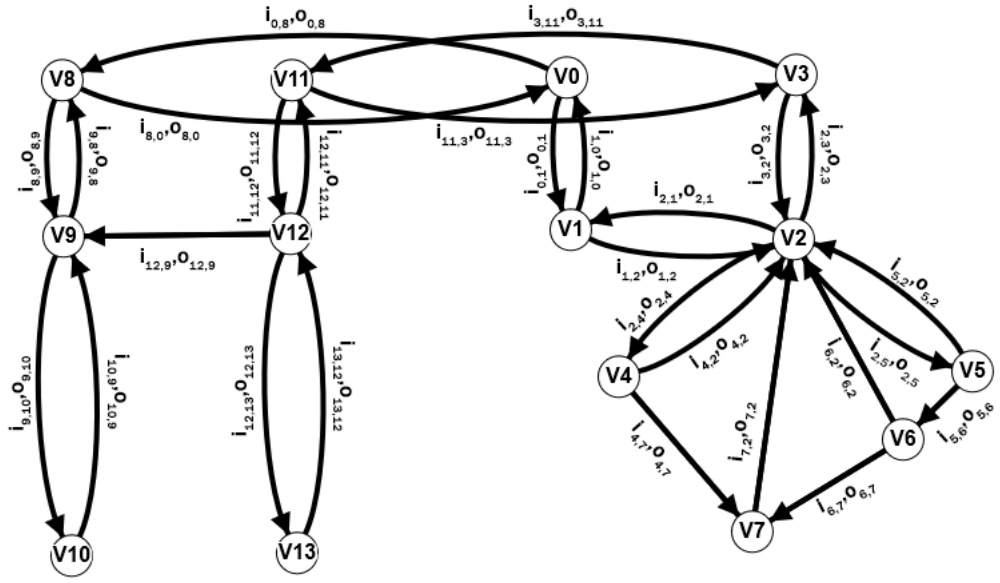


Рисунок 4.1 – Граф $G_{m_{амдз}}$ для СЗИ НСД «Аккорд-АМДЗ».

СЗИ (как в разработанной программе тестирования из раздела 3.5), а также использовать разработанный в разделе 3.2 алгоритм тестирования программно-аппаратных СЗИ с применением средств виртуализации. В соответствии с чем, если каким-либо образом автоматизировать эти переходы, то $m_{амдз} \in M_a$. В противном случае – $m_{амдз} \in M_{па}$.

Однако, как было показано в разделе 3.3, автоматическое выполнение таких переходов на физических СВТ является труднореализуемой задачей и требует применения специализированных промышленных автоматизированных комплексов, масштабность, сложность и стоимость которых не позволит использовать их в процессе тестирования большинству производителей программно-аппаратных СЗИ. В соответствии с этим полную автоматизацию тестирования «Аккорд-АМДЗ» можно выполнить только с использованием средств виртуализации из раздела 3.2, но, возможно, с ухудшением «чистоты» тестирования.

Вторым шагом предложенного в Главе 2 способа тестирования программно-аппаратных СЗИ является проведение тестирования при помощи алгоритма решения задачи тестирования с применением теории графов. Согласно данному алгоритму задача тестирования будет иметь решение тогда и только тогда, когда любая вершина $v \in V$ производного для $G_{m_{амдз}}$ графа $G'_{m_{амдз}}$ либо принадлежит орцепи, либо лежит в компоненте сильной связностости. При этом в соответствии с Утверждением 1 из раздела 2.3, если аналогичное условие выполняется для графа $G_{m_{амдз}}$, то задача тестирования также имеет решение.

Для того, чтобы показать, что все вершины графа $G_{m_{амдз}}$ в соответствии с шагом 3 алгоритма решения задачи тестирования из раздела 2.3 либо принадлежат орцепи, либо лежат в компоненте сильной связностости, применим алгоритм Косараджу–Шарира и выполним два обхода графа в глубину. Результаты обходов графа в глубину представлены на Рисунках 4.2 и 4.3.

Таким образом, граф $G_{m_{амдз}}$ является сильно связным, то есть для СЗИ НСД «Аккорд-АМДЗ» может быть применен алгоритм решения задачи тестирования с использованием теории графов из Главы 2, и задача тестирования его функций безопасности может быть решена. Помимо это-

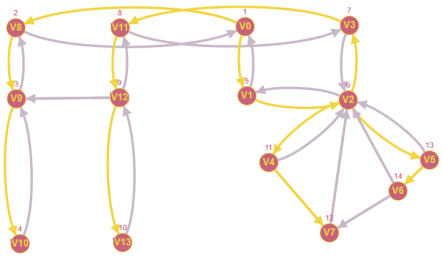


Рисунок 4.2 – Результат обхода графа $G_{m_{амдз}}$ в глубину из начальной вершины v_0, все вершины достижимы из v_0 – граф $G_{m_{амдз}}$, по крайней мере, слабо связный.

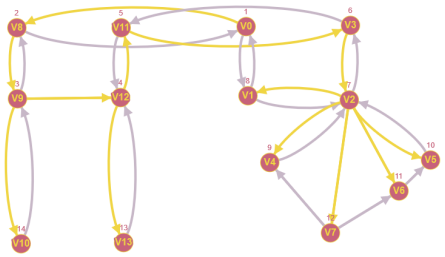


Рисунок 4.3 – Результат обхода обратного графа $G_{m_{амдз}}$ в глубину из начальной вершины v_0, все вершины достижимы из v_0 – граф $G_{m_{амдз}}$ сильно связный.

го в соответствии с Определением 2 из Главы 2 будет обеспечена полнота и оптимальность тестирования рассматриваемого СЗИ. Также необходимо отметить, что при решении задачи тестирования для исходного графа $G_{m_{амдз}}$, а не для его производного графа $G'_{m_{амдз}}$, в соответствии с выводами раздела 2.3 может быть решена не только задача функционального тестирования, но и задача негативного тестирования данного средства защиты.

Аналогично предыдущему СЗИ, основываясь на результатах Главы 2 обозначим ПСКЗИ ШИПКА как $m_{шипка} \in M$. В соответствии с документацией на ПСКЗИ ШИПКА [66] – $m_{шипка} \in P_{ос}$ и $m_{шипка} \in P_{моб}$. Представим ПСКЗИ ШИПКА в виде конечного детерминированного автомата $\widetilde{m}_{шипка} = (V_{шипка}, I_{шипка}, O_{шипка}, f_{шипка}, g_{шипка})$.

Согласно документации на ПСКЗИ ШИПКА [66] $m_{шипка}$ может находиться в следующих состояниях (для упрощения восприятия рассматриваемых состояний далее опустим обозначение *«шипка»* при их описании) – $v_0, v_1, ..., v_{22} \in V_{шипка}$, когда устройство:

v_0 – не подключено (не установлено в USB-порт ПК) и не инициализированно (не установлены параметры авторизации, не выполнено форматирование, не задан PIN-код пользователя) – начальное состояние СЗИ;

v_1 – подключено (установлено в USB-порт) и не инициализированно;

v_2 – подключено и инициализированно (установлены параметры авторизации, выполнено форматирование, задан PIN-код пользователя);

v_3 – выполнена идентификация/аутентификация пользователя (введен PIN-код);

v_4 – сгенерированы криптографические ключи;

v_5 – выполнена криптографическая операция (зашифрование/расшифрование/вычисление электронной подписи/проверка электронной подписи файла);

v_6 – отключено (изъято из USB-порта ПК), имеются криптографические ключи;

v_7 – подключено в USB-порт ПК, имеются криптографические ключи;

v_8 – выполнена идентификация/аутентификация пользователя (введен PIN-код), имеются криптографические ключи;

v_9 – создана привязка пользователя к учетной записи в ОС ПК;

v_{10} – выполнена идентификация/аутентификация пользователя в ОС ПК;

v_{11} – отключено, имеется привязка пользователя к учетной записи в ОС ПК;

v_{12} – подключено, имеется привязка пользователя к учетной записи в ОС ПК;

v_{13} – выполнена идентификация/аутентификация пользователя (введен PIN-код), имеется привязка пользователя к учетной записи в ОС ПК;

v_{14} – отключено, произошла блокировка сессии пользователя;

v_{15} – находится в состоянии аналогичном v_4 и v_9 одновременно;

v_{16} – находится в состоянии аналогичном v_5 и v_9 одновременно;

v_{17} – находится в состоянии аналогичном v_6 и v_9 одновременно;

v_{18} – находится в состоянии аналогичном v_7 и v_9 одновременно;

v_{19} – находится в состоянии аналогичном v_8 и v_9 одновременно;

v_{20} – находится в состоянии аналогичном v_{10} и v_4 одновременно;

v_{21} – находится в состоянии аналогичном v_{14} и v_4 одновременно;

v_{22} – инициализировано и отключено.

Используя результаты Главы 2 построим граф $G_{m_{шипка}} = (V_{шипка}, E_{шипка})$, соответствующий конечному автомату $\widetilde{m}_{шипка} = (V_{шипка}, I_{шипка}, O_{шипка}, f_{шипка}, g_{шипка})$, на котором множества автомата $I_{шипка}, O_{шипка}, T_{шипка}$ представим в виде дуг, см. Рисунок 4.4.

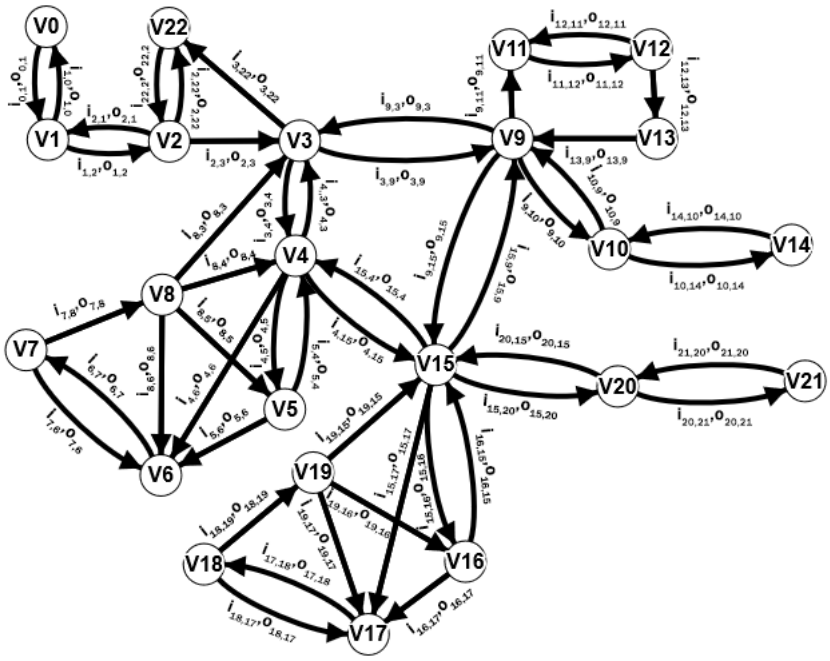


Рисунок 4.4 – Граф $G_{m_{шипка}}$ для ПСКЗИ ШИПКА.

В соответствии с документацией [66] все переходы из одного состояния в другое с помощью стимулов $I_{шипка}$ могут быть выполнены вручную (некоторые с помощью ручного отключения и подключения СЗИ в ПК), то есть $m_{шипка} \in M_p$.

Все переходы $T_{шипка}$, соответствующие дугам графа, кроме (v_0, v_1), (v_1, v_0), (v_2, v_{22}), (v_{22}, v_2), (v_3, v_{22}), (v_5, v_6), (v_4, v_6), (v_6, v_7), (v_7, v_6), (v_8, v_6), (v_{16}, v_{17}), (v_{15}, v_{17}), (v_{17}, v_{18}), (v_{18}, v_{17}), (v_{19}, v_{17}), (v_9, v_{11}), (v_{11}, v_{12}), (v_{12}, v_{11}), (v_{10}, v_{14}), (v_{14}, v_{10}), (v_{20}, v_{21}) и (v_{21}, v_{20}) (подключение/отключение устройства в USB-порт ПК) могут быть выполнены автоматически в том случае, если использовать средства автоматизации и программы тестирования в ОС СВТ (как в разработанной программе тестирования из раздела 3.5), а также использовать разработанный в разделе 3.2 ал-

горитм тестирования программно-аппаратных СЗИ с применением средств виртуализации для функций безопасности, выполняющихся до старта сессии пользователя. Перечисленные же переходы можно осуществлять с использованием разработанного вспомогательного программно-технического средства тестирования, способного автоматически подключать и отключать ПСКЗИ ШИПКА от USB-порта СВТ, например, разработанного в разделе 3.4 коммутатора USB-канала. В соответствии с этим $m_{шипка} \in M_a$. В противном случае – $m_{шипка} \in M_{па}$.

Согласно алгоритму решения задачи тестирования с использованием теории графов из Главы 2, она будет иметь решение тогда и только тогда, когда когда любая вершина $v \in V$ производного для $G_{m_{шипка}}$ графа $G'_{m_{шипка}}$ либо принадлежит орцепи, либо лежит в компоненте сильной связностости. При этом в соответствии с Утверждением 1 из раздела 2.3, аналогичное условие выполняется для графа $G_{m_{шипка}}$, то задача тестирования также имеет решение.

Для того, чтобы показать, что все вершины графа $G_{m_{шипка}}$ в соответствии с шагом 3 алгоритма решения задачи тестирования из раздела 2.3 либо принадлежат орцепи, либо лежат в компоненте сильной связностости, применим алгоритм Косараджу–Шарира и выполним два обхода графа в глубину. Результаты обходов графа в глубину представлены на Рисунках 4.5 и 4.6.

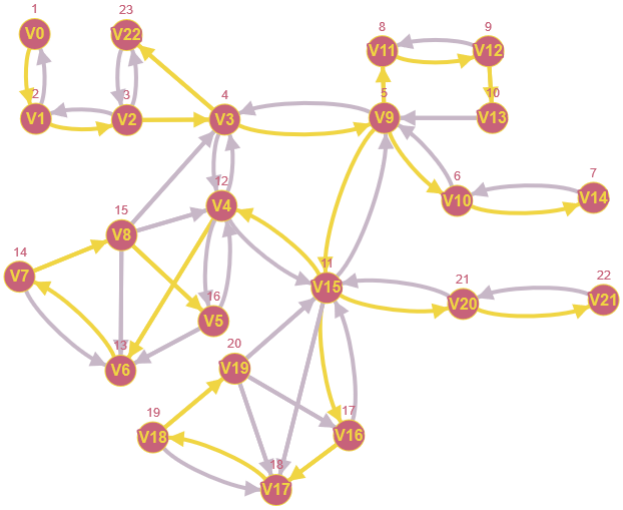


Рисунок 4.5 – Результат обхода графа $G_{m_{шипка}}$ в глубину из начальной вершины v_0, все вершины достижимы из v_0 – граф $G_{m_{шипка}}$, по крайней мере, слабо связный.

Граф $G_{m_{шипка}}$ является сильно связным, то есть для ПСКЗИ ШИПКА может быть применен алгоритм решения задачи тестирования с использованием теории графов из Главы 2, и задача тестирования его функций безопасности может быть решена. Помимо этого в соответствии с Определением 2 из Главы 2 будет обеспечена полнота и оптимальность тестирования рассматриваемого СЗИ. Также необходимо отметить, что при решении задачи тестирования для исходного графа $G_{m_{шипка}}$, а не для его производного графа $G'_{m_{шипка}}$, в соответствии с выводами раздела 2.3 может быть решена не только задача функционального тестирования, но и задача негативного тестирования данного средства защиты.

Необходимо отметить, что разработанные в рамках раздела 3.5 программы тестирования учитывают все перечисленные состояния рассматриваемых СЗИ, и с применением вспомога-

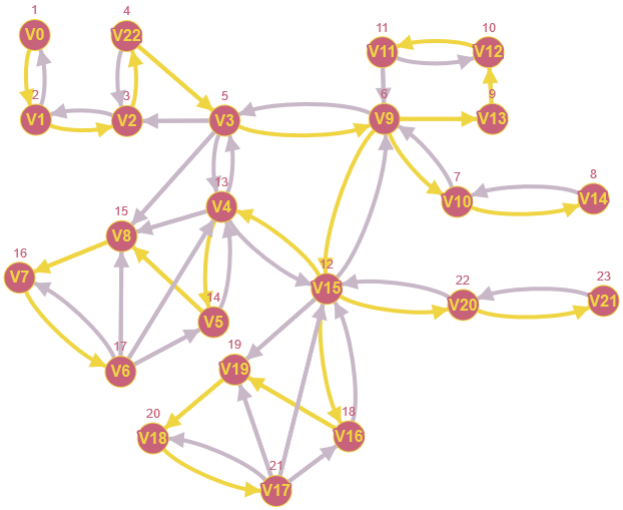


Рисунок 4.6 – Результат обхода обратного графа $G_{m_{шипка}}$ в глубину из начальной вершины v_0, все вершины достижимы из v_0 – граф $G_{m_{шипка}}$ сильно связный.

тельных средств и предложенных алгоритмов позволяют полностью автоматизировать процесс тестирования этих средств защиты информации.

Третьим шагом предложенного в Главе 2 способа тестирования программно-аппаратных СЗИ является верификация с использованием теории оптимизации и принятия решений. В Главе 3 были приведены рекомендации по разработке программы верификации программно-аппаратных СЗИ, основанной на результатах Главы 2. С использованием этих рекомндеций была разработана программа «Верификатор программно-аппаратных СЗИ», реализующая базовый и расширенный режимы работы. Используем поочередно оба режима программы для проведения верификации ПСКЗИ ШИПКА (для СЗИ НСД «Аккорд-АМДЗ» применение данной программы будет выглядеть аналогично, поэтому рассмотрение верификации данного СЗИ опустим).

Предположим, что необходимо произвести верификацию очередной версии программно-аппаратного СЗИ (например, v3.6.0.0) на основе выявленных в работе программ тестирования ошибок. Для сравнения в программе верификации должны использоваться результаты тестирования предыдущих версий (например, v3.3.0.0, v3.4.0.0 и v3.5.0.0) и некой «эталонной» версии СЗИ. При этом, предположим, что в ходе работы программ тестирования для указанных версий СЗИ зафиксировано следующее количество влияющих на защищенность ИС критических ошибок, ошибок в функциях безопасности и некритических ошибок – необходимо задать их в виде вектора, значения элементов которого соответствуют количеству перечисленных видов ошибок соответственно, например, v3.3.0.0 – (1, 1, 1); v3.4.0.0 – (2, 1, 1); v3.5.0.0 – (0, 1, 5); v3.6.0.0 – (0, 0, 10). Пример результатов такой верификации заданных ранее версий для ПСКЗИ ШИПКА приведен на Рисунке 4.9 (вместо значений весовых коэффициентов для удобства указывается процентное отношение) [105].

Из полученных результатов видно, что ПСКЗИ ШИПКА v3.6.0.0 имеет наиболее близкое к «эталонной» версии значение обобщенного критерия (см. раздел 2.4), а также наименьшую из всех остальных версий критичность выявленных ошибок. Это говорит о положительной тенденции при проведении регрессионного тестирования – по сравнению с предыдущими версиями

Верификатор программно-аппаратных СЗИ v1.0 | Входные данные | Визуализация данных | Анализ результата

	Важность	ПСКЗИ ШИПКА v3.3.0.0	ПСКЗИ ШИПКА v3.4.0.0	ПСКЗИ ШИПКА v3.5.0.0	ПСКЗИ ШИПКА v3.6.0.0	Эталонный вариант
Оценка критичности ошибок в функциях безопасности СЗИ	100.0%	17.5%	46.3%	13.7%	12.6%	10.0%
Влияющие на защищенность ИС	65.5%	10.4%	39.3%	5.3%	5.3%	5.3%
Нарушающие работу функций безопасности	29.0%	6.7%	6.7%	6.7%	4.5%	4.5%
Несущественные ошибки	5.5%	0.3%	0.3%	1.7%	2.9%	0.2%

Рисунок 4.7 – Результаты верификации ПСКЗИ ШИПКА на основе ошибок, выявленных при работе программ тестирования (базовый режим)

количество ошибок СЗИ выросло, при этом общая их критичность уменьшилась. Таким образом, верификацию ПСКЗИ ШИПКА v3.6.0.0 можно завершить, а выявленные ошибки учесть при разработке уже следующей версии СЗИ. Также в результатах верификации на Рисунке 4.9 видно, что версия ПСКЗИ ШИПКА v3.4.0.0 вследствие большего количества ошибок, влияющих на защищенность ИС или данных, имеет наибольшую критичность выявленных ошибок. В соответствии с этим верификация данной версии была прервана в момент проведения, а СЗИ отправлено на доработку. Результатом доработки стала v3.5.0.0, в которой были исправлены все наиболее критические ошибки, но допущены несколько новых ошибок наименьшего уровня критичности. Однако вследствие того, что общая критичность всех выявленных ошибок сократилась, в том числе по сравнению с версией v3.3.0.0, верификация ПСКЗИ ШИПКА v3.5.0.0 была успешно завершена [33, 105].

Необходимо пояснить, что значения обобщенного и частных критериев (по конкретным видам ошибок) для «эталонной» версии СЗИ на Рисунке 4.9 имеют ненулевое значение посредством использованного для решения задачи математического аппарата и принятой в нем шкалы превосходства, даже с учетом нулевых количественных показателей всех критериев оценки. Приведенные значения критериев для «эталонной» версии необходимо трактовать как минимально допустимые значения, которые потенциально не могут быть превышены в любой реальной версии программно-аппаратного СЗИ.

При этом наличие ошибок в идентификации и аутентификации для доступа к функциям безопасности, а также в экспорте/импорте закрытых и симметричных ключей будет иметь наибольший приоритет, в генерации ключей – приоритет ниже, в зашифровании/расшифровании и генерации/проверке электронной подписи – еще ниже, а в экспорте и импорте открытых ключей приоритет минимальный. Полученные ранее отношения «важности» (превосходства) для самих видов ошибок (выявленных в соответствующей функции безопасности), а также шкала критичности могут быть использованы без изменений. В соответствии с этим в расширенном режиме работы программы «Верификатор программно-аппаратных СЗИ» для тех же входных данных были получены результаты, представленные на Рисунке 4.10.

В данном режиме работы результаты верификации ПСКЗИ ШИПКА другие [33, 105]:

	Важность	ПСКЗИ ШИПКА v3.3.0.0	ПСКЗИ ШИПКА v3.4.0.0	ПСКЗИ ШИПКА v3.5.0.0	ПСКЗИ ШИПКА v3.6.0.0	Эталонный вариант
Оценка критичности ошибок в функциях безопасности СЗИ	100.0%	20.7%	20.9%	19.6%	20.2%	18.6%
И/А для доступа к ФБ	23.8%	6.2%	4.2%	4.2%	5.0%	4.2%
Генерация ключевой информации	19.0%	3.8%	3.8%	3.8%	3.9%	3.8%
Шифрование и расшифрование	14.3%	2.8%	2.8%	3.2%	2.8%	2.8%
Подпись и проверка подписи	14.3%	2.7%	3.1%	3.2%	2.7%	2.7%
Экспорт и импорт открытых ключей	4.8%	0.5%	2.5%	0.7%	0.5%	0.5%
Экспорт и импорт закрытых ключей	23.8%	4.7%	4.6%	4.6%	5.3%	4.6%

Рисунок 4.8 – Результаты верификации ПСКЗИ ШИПКА на основе ошибок и различного уровня критичности функций безопасности, в которых они зафиксированы

1. Общая критичность ошибок версий v3.3.0.0 и v.3.4.0.0 не так сильно отличаются между собой посредством того, что в v3.4.0.0 все критические ошибки были выявлены в наименее важной функции безопасности. Возможно, что при таких результатах верификацию этой версии ПСКЗИ ШИПКА можно было бы завершить посредством того, что по сравнению с v3.3.0.0 были исправлены критические ошибки в более значимой функции безопасности.
2. Версия v3.5.0.0 имеет меньшую общую критичность ошибок, чем v3.6.0.0 посредством того, что в последней было выявлено множество незначительных ошибок в двух наиболее значимых функциях безопасности. СЗИ v3.6.0.0 в таких обстоятельствах, возможно, было бы целесообразней отправить на доработку.

Пример результатов такой верификации заданных ранее версий для ПСКЗИ ШИПКА приведен на Рисунке 4.9 (вместо значений весовых коэффициентов для удобства указывается процентное отношение) [105].

Верификатор программно-аппаратных СЗИ v1.0 | Входные данные | Визуализация данных | Анализ результата

	Важность	ПСКЗИ ШИПКА v3.3.0.0	ПСКЗИ ШИПКА v3.4.0.0	ПСКЗИ ШИПКА v3.5.0.0	ПСКЗИ ШИПКА v3.6.0.0	Эталонный вариант
Оценка критичности ошибок в функциях безопасности СЗИ	100.0%	17.5%	46.3%	13.7%	12.6%	10.0%
Влияющие на защищенность ИС	65.5%	10.4%	39.3%	5.3%	5.3%	5.3%
Нарушающие работу функций безопасности	29.0%	6.7%	6.7%	6.7%	4.5%	4.5%
Несущественные ошибки	5.5%	0.3%	0.3%	1.7%	2.9%	0.2%

Рисунок 4.9 – Результаты верификации ПСКЗИ ШИПКА на основе ошибок, выявленных при работе программ тестирования (базовый режим)

Из полученных результатов видно, что ПСКЗИ ШИПКА v3.6.0.0 имеет наиболее близкое к «эталонной» версии значение обобщенного критерия (см. раздел 2.4), а также наименьшую из всех остальных версий критичность выявленных ошибок. Это говорит о положительной тенденции при проведении регрессионного тестирования – по сравнению с предыдущими версиями количество ошибок СЗИ выросло, при этом общая их критичность уменьшилась. Таким образом, верификацию ПСКЗИ ШИПКА v3.6.0.0 можно завершить, а выявленные ошибки учесть при разработке уже следующей версии СЗИ. Также в результатах верификации на Рисунке 4.9 видно, что версия ПСКЗИ ШИПКА v3.4.0.0 вследствие большего количества ошибок, влияющих на защищенность ИС или данных, имеет наибольшую критичность выявленных ошибок.

В соответствии с этим верификация данной версии была прервана в момент проведения, а СЗИ отправлено на доработку. Результатом доработки стала v3.5.0.0, в которой были исправлены все наиболее критические ошибки, но допущены несколько новых ошибок наименьшего уровня критичности. Однако вследствие того, что общая критичность всех выявленных ошибок сократилась, в том числе по сравнению с версией v3.3.0.0, верификация ПСКЗИ ШИПКА v3.5.0.0 была успешно завершена [33, 105].

Необходимо пояснить, что значения обобщенного и частных критериев (по конкретным видам ошибок) для «эталонной» версии СЗИ на Рисунке 4.9 имеют ненулевое значение посредством использованного для решения задачи математического аппарата и принятой в нем шкалы превосходства, даже с учетом нулевых количественных показателей всех критериев оценки. Приведенные значения критериев для «эталонной» версии необходимо трактовать как минимально допустимые значения, которые потенциально не могут быть превышены в любой реальной версии программно-аппаратного СЗИ.

При этом наличие ошибок в идентификации и аутентификации для доступа к функциям безопасности, а также в экспорте/импорте закрытых и симметричных ключей будет иметь наибольший приоритет, в генерации ключей – приоритет ниже, в зашифровании/расшифровании и генерации/проверке электронной подписи – еще ниже, а в экспорте и импорте открытых ключей приоритет минимальный. Полученные ранее отношения «важности» (превосходства) для самих видов ошибок (выявленных в соответствующей функции безопасности), а также шкала критичности могут быть использованы без изменений. В соответствии с этим в расширенном режиме работы программы «Верификатор программно-аппаратных СЗИ» для тех же входных данных были получены результаты, представленные на Рисунке 4.10.

	Важность	ПСКЗИ ШИПКА v3.3.0.0	ПСКЗИ ШИПКА v3.4.0.0	ПСКЗИ ШИПКА v3.5.0.0	ПСКЗИ ШИПКА v3.6.0.0	Эталонный вариант
Оценка критичности ошибок в функциях безопасности СЗИ	100.0%	20.7%	20.9%	19.6%	20.2%	18.6%
И/А для доступа к ФБ	23.8%	6.2%	4.2%	4.2%	5.0%	4.2%
Генерация ключевой информации	19.0%	3.8%	3.8%	3.8%	3.9%	3.8%
Шифрование и расшифрование	14.3%	2.8%	2.8%	3.2%	2.8%	2.8%
Подпись и проверка подписи	14.3%	2.7%	3.1%	3.2%	2.7%	2.7%
Экспорт и импорт открытых ключей	4.8%	0.5%	2.5%	0.7%	0.5%	0.5%
Экспорт и импорт закрытых ключей	23.8%	4.7%	4.6%	4.6%	5.3%	4.6%

Рисунок 4.10 – Результаты верификации ПСКЗИ ШИПКА на основе ошибок и различного уровня критичности функций безопасности, в которых они зафиксированы

В данном режиме работы результаты верификации ПСКЗИ ШИПКА другие [33, 105]:

1. Общая критичность ошибок версий v3.3.0.0 и v.3.4.0.0 не так сильно отличаются между собой посредством того, что в v3.4.0.0 все критические ошибки были выявлены в наименее важной функции безопасности. Возможно, что при таких результатах верификацию этой версии ПСКЗИ ШИПКА можно было бы завершить посредством того, что по сравнению с v3.3.0.0 были исправлены критические ошибки в более значимой функции безопасности.
2. Версия v3.5.0.0 имеет меньшую общую критичность ошибок, чем v3.6.0.0 посредством того, что в последней было выявлено множество незначительных ошибок в двух наиболее зна-

чимых функциях безопасности. СЗИ v3.6.0.0 в таких обстоятельствах, возможно, было бы целесообразней отправить на доработку.

Таким образом, подтверждена практическая возможность использования результатов из Главы 2 и 3 к функциям безопасности программно-аппаратных СЗИ.

Предложенный в Главе 2 способ тестирования функций безопасности программно-аппаратных СЗИ, а также сформулированные в разделе 3.3 требования к средствам тестирования и среде их применения позволяют проводить тестирование на более качественном уровне. В отличие от используемых ранее существующих способов тестирования ПО, стало возможным обеспечить полноту тестирования, а также применять средства тестирования и вследствие этого понизить сложность его проведения (например, нагрузочного тестирования).

Возможность практического использования предложенного в Главе 2 способа тестирования программно-аппаратных СЗИ и входящих в его состав алгоритмов тестирования и верификации их функций безопасности, а также сформулированных в разделе 3.3 требований к средствам тестирования таких средств защиты при автоматическом тестировании подтверждается реализацией на их основе комплекса программ «Тестирование функций безопасности программно-аппаратных средств защиты информации» (свидетельство о государственной регистрации программы для ЭВМ №2016616332, см. Приложение Г [80]), позволяющего выполнять автоматизированное тестирование различных видов функций безопасности программно-аппаратных СЗИ. Помимо этого в соответствии с требованиями и рекомендациями по практической реализации разработано вспомогательное средство тестирования – коммутатор USB-канала, при использовании которого совместно с программами тестирования становится возможным автоматически тестировать функции безопасности СЗИ, реализованные на базе мобильной аппаратной компоненты.

Положенные в основу программ тестирования способ из раздела 2.5 и требования из раздела 3.3 сформулированы с учетом обеспечения полноты и оптимальности проводимого тестирования, при котором проверка каждой функции безопасности проводится во всех состояниях, в которых она должна выполняться. На основании этого можно сделать вывод о полноте и оптимальности тестирования, выполняемого с использованием разработанного комплекса программ.

4.2. Анализ результатов применения разработанного программно-аппаратного комплекса для тестирования СЗИ

В Главе 3 разработаны программы тестированиия и верификации для ПСКЗИ ШИПКА и ПАК СЗИ НСД «Аккорд-Х», основанные на предложенном в Главе 2 способе тестирования функций безопасности программно-аппаратных СЗИ. Применение таких программ позволяет исключить «человеческий фактор», неточности в проведении тестирования, а также сократить временные затраты на процесс тестирования и верификации. Был проведен ряд экспериментов, подтверждающих последнее утверждение.

Вначале с использованием разработанных ПМИ и предложенного в разделе 2.3 алгоритма тестирования, а также требований из раздела 3.3 было проведено несколько экспериментов ручного тестирования функций безопасности различных версий выбранных программно-

аппаратных СЗИ. При этом тестирование поочередно проводилось на различных ОС, в которых, согласно документации на комплексы, может функционировать каждое из этих СЗИ, а также – с различными исполнениями аппаратных компонент, реализующих функции безопасности рассматриваемых средств защиты. При проведении каждой проверки было измерено затраченное на нее время, а также время анализа полученных результатов. На основании среднего значения времени, затраченного на проведение ручного тестирования в каждой из поддерживаемых ОС со всеми вариантами исполнений аппаратной компоненты, а также времени анализа его результатов, было рассчитано общее суммарное время ручного тестирования для каждого из рассматриваемых средств защиты. Также в каждом эксперименте на основании полученных при тестировании ошибок и предложенного алгоритма из раздела 2.4 была выполнена верификация СЗИ, измерено время ее ручного проведения (не включающее время, затраченное на тестирование) и получено среднее значение.

Затем было проведено автоматическое тестирование функций безопасности выбранных программно-аппаратных СЗИ с использованием предложенного алгоритма тестирования из раздела 2.3 и разработанных программ тестирования (во всех ОС, со всеми исполнениями аппаратной компоненты). После чего рассчитано общее суммарное время данного автоматического тестирования (рассчитывается с учетом параллельного выполнения программ тестирования и анализа результатов их работы) и время на проведение верификации с использованием соответствующей программы из разработанного комплекса. Измеренные и рассчитанные показатели занесены в Таблицу 4.1 [31].

Необходимо отметить, что в данной таблице временные характеристики для ручного тестирования рассчитываются в человеко-часах (чел.-ч.), а для автоматического – в часах (ч.), кроме времени анализа результатов автоматических проверок. Помимо этого при переводе затраченного на ручное тестирование времени из чел.-час в чел.-дни необходимо учитывать, что рабочий день обычно составляет 8 часов (то есть 1 чел.-день соответствует 8 чел.-часам). Такого ограничения нет при расчете временных характеристик для автоматического тестирования – программы тестирования могут работать и без участия тестировщика, то есть и в нерабочее время. Таким образом, общее затраченное время при переводе из часов в дни для ручного и автоматического тестирования рассчитывается по-разному. При этом необходимо учитывать, что анализ результатов автоматического тестирования может выполняться параллельно с функционированием программ тестирования.

Таблица 4.1 – Результаты экспериментальных исследований: временные характеристики ручного и автоматического тестирования, верификации программно-аппаратных СЗИ

Характеристики, показатели экспериментальных исследований	**ПСКЗИ ШИПКА**	**«Аккорд-Х»**
1. Количество ОС	10	15
2. Количество аппаратных компонент	6	6
3. Время РТ с анализом, чел.-ч. (1 ОС, 1 апп. компонента)	20	16

Продолжение таблицы 4.1

Характеристики, показатели экспериментальных исследований	**ПСКЗИ ШИПКА**	**«Аккорд-Х»**
4. Общее время РТ, чел.-ч.	1200	1440
5. ***Общее время РТ, чел.-д.***	*150*	*180*
6. ***Время РВ, чел.-ч.***	*16*	*25*
7. Время АТ (без анализа результатов), ч. (1 ОС, 1 апп. компонента)	2,5	1,5
8. Общее время АТ, ч.	150	135
9. Время анализа результатов АТ, чел.-ч. (1 ОС, 1 апп. компонента)	2,5	3
10. Общее время анализа результатов АТ, чел.-ч.	150	270
11. ***Общее время АТ, чел.-д.***	*18, 75*	*33,75*
12. ***Время АВ, ч.***	*0,04*	*0,08*

Где:

РТ – ручное тестирование;

АТ – автоматическое тестирование;

РВ – ручная верификация;

АВ – автоматическая верификация.

Как видно из Таблицы 4.1, временные затраты на ручное тестирование и верификацию одной версии любого из рассматриваемых средств защиты являются достаточно большими – для их проведения во всех возможных ОС со всеми возможными вариантами исполнения аппаратной компоненты в среднем необходимо более полугода работы одного тестировщика, из которых ручная верификация займет не менее двух рабочих дней. При этом важным является тот факт, что временные затраты на тестирование и верификацию включаются во время, затрачиваемое на внедрение СЗИ в ИС, а это значит, что в рассматриваемом случае внедрение будет также продолжаться не менее полугода. Кроме того, в процессе внедрения средства защиты в ИС может быть найден ряд ошибок, влияющих на пользовательские и функциональные характеристики данной системы. На основании этого будет необходимо вносить изменения в СЗИ с последующей сборкой его очередной версии, также требующей тестирования и верификации, а предыдущие их результаты окажутся неактуальными. Тем самым повторное тестирование и верификация увеличат срок внедрения еще на полгода. В соответствии с этим проведение ручных проверок с учетом достаточно быстрого изменения условий тестирования (выхода обновлений для ИС, ОС и тому подобное) приводит к неадекватным срокам внедрения СЗИ в ИС [34].

В случае применения предложенного способа, и входящих в него алгоритмов, на автоматическое тестирование затрачивается не более одного-двух месяцев, а результаты автоматической верификации можно получить в течение нескольких минут (при условии, что программы тестирования выдают ошибки с уже проставленным рангом критичности, как это было рассмотрено в разделе 3.5), что уже представляет собой адекватные сроки.

В результате за время одного цикла полного ручного тестирования программно-аппаратного СЗИ со всеми возможными сочетаниями ОС и исполнений аппаратной компоненты, при автоматическом тестировании с аналогичными условиями уже будут исправлены все выявленные ошибки, произведены повторное тестирование и верификация, а также завершено внедрение в ИС. Аналогичная ситуация будет складываться в случае, когда для внедрения в ИС нет необходимости тестировать все возможные сочетания ОС и исполнений аппаратной компоненты (временные характеристики будут другими, но их соотношение между собой не изменится).

Изложенные выше аспекты подтверждают положительный эффект от применения разработанных программ тестирования, а также используемого при тестировании предложенного способа тестирования функций безопасности программно-аппаратных СЗИ.

Сравнение результатов ручного и автоматического тестирования с верификацией приведено на диаграмме Рисунка 4.11 [32].

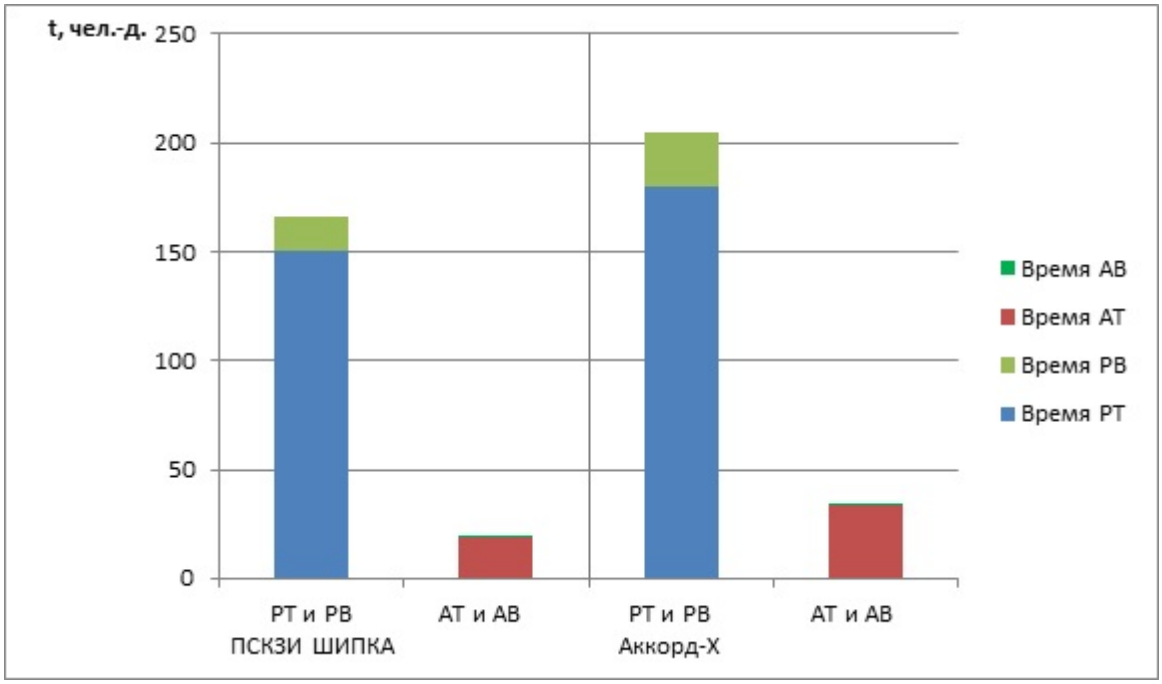


Рисунок 4.11 – Гистограмма временных затрат на ручное и автоматическое тестирование/верификацию средств защиты, реализующих функции безопасности различных видов

Также при применении предложенного способа и средств тестирования для функций безопасности программно-аппаратных СЗИ различных видов показано количественное улучшение результатов (см. Рисунок 4.12) в части уменьшения непроводимых ранее проверок функций безопасности и увеличения общего числа проверок путем учета всех возможных переходов (как следствие обеспечения полноты тестирования).

Выявлено, что до применения предложенного способа и средств тестирования функций безопасности программно-аппаратных СЗИ количество непроводимых проверок для ПСКЗИ ШИПКА и ПАК СЗИ НСД «Аккорд-Х» составляло примерно 25-30% и 20% от общего числа проверок соответственно. В результате применения данного способа и разработанного вспомогательного средства тестирования коммутатора USB-канала непроводимые ранее проверки

для ПСКЗИ ШИПКА удалось исключить вообще, а для «Аккорд-X» – минимизировать до 5% (посредством отсутствия средства автоматизированного встраивания аппаратной компоненты в СВТ). При этом, вследствие учета всех возможных переходов, для обоих средств защиты общее количество проверок увеличилось примерно на 10%.

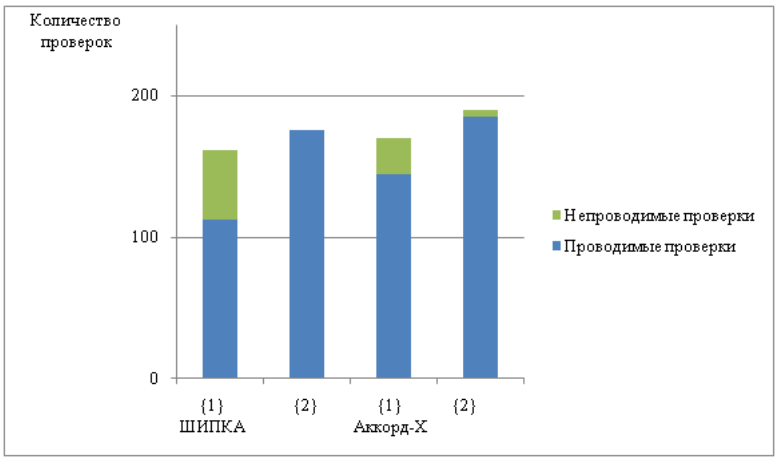


Рисунок 4.12 – Изменение количества и соотношения непроводимых и проводимых проверок функций безопасности программно-аппаратных СЗИ ({1} и {2} – без и с применением предложенного способа и разработанных средств тестирования)

Внедрение результатов работы

В диссертационном исследовании представлены разработанные автором модель СЗИ, критерии применимости существующих способов тестирования ПО к функциям безопасности программно-аппаратных СЗИ, алгоритмы их тестирования и верификации, способ, а также требования к средствам тестирования различных видов функций безопасности и среде их применения. На основе данных требований разработан программный комплекс «Тестирование функций безопасности программно-аппаратных средств защиты информации» (свидетельство о государственной регистрации программы для ЭВМ №2016616332). Применение данного комплекса совместно с разработанным в рамках данных исследований коммутатором USB-канала позволяет выполнять автоматическое тестирование различных видов функций безопасности программно-аппаратных СЗИ [32, 104].

Разработанный комплекс программ применяется в ЗАО «ОКБ САПР» для тестирования функций безопасности следующих программно-аппаратных СЗИ при внедрении их в ИС (см. раздел 1.2):

– ПСКЗИ ШИПКА – персональное средство криптографической защиты;

– ПАК СЗИ НСД «Аккорд-X» – программно-аппаратное средство разграничения доступа пользователей в ОС.

Программы тестирования для ПСКЗИ ШИПКА, входящие в комплекс «Тестирование функций безопасности программно-аппаратных средств защиты информации», используются совместно с разработанным вспомогательным средством тестирования – коммутатором USB-канала,

что позволяет выполнять автоматическое подключение/отключение данного СЗИ к/от СВТ в процессе его тестирования.

Программы тестирования для ПАК СЗИ НСД «Аккорд-X», также входящие в разработанный комплекс, применяются совместно со средствами виртуализации, которые позволяют осуществить перенаправление аппаратной компоненты данного средства защиты в ОС ВМ. При этом используется предложенный алгоритм тестирования функций безопасности программно-аппаратных СЗИ с использованием средств виртуализации.

Программы тестирования «Аккорд-X» также содержат возможность автоматической сборки подсистемы разграничения доступа для различных поддерживаемых ОС. Такая сборка выполняется до автоматической установки «Аккорд-X» в ОС и запуска проверок его функций безопасности. В связи с этим становится возможным организовать регрессионное тестирование, при котором любые изменения в исходных кодах подсистемы разграничения доступа автоматически проверяются на предмет нарушения корректно работающих ранее функций безопасности.

Кроме того, в процессе верификации рассматриваемых СЗИ используются программы тестирования, входящие в разработанный комплекс, и реализующие предложенный алгоритм верификации программно-аппаратных СЗИ, основанный на классификации обнаруженных ошибок, анализе степени их критичности и влияния на защищенность системы.

Предложенный способ тестирования функций безопасности программно-аппаратных СЗИ применяется в процессе разработки как уже рассмотренных ПСКЗИ ШИПКА и «Аккорд-X», так и других средств защиты, выпускаемых данной организацией.

Помимо этого, результаты проведенных автором исследований, а именно модель СЗИ, критерии применимости существующих способов тестирования к функциям безопасности программно-аппаратных СЗИ, алгоритмы и способ тестирования, а также требования к средствам тестирования функций безопасности и среде их применения, используются «ОКБ САПР» в рамках разработки перспективных средств тестирования для других средств защиты.

Результаты диссертационного исследования успешно применяются в ФАУ «ГНИИИ ПТЗИ ФСТЭК России» при разработке программ и методик сертификационных испытаний, а также при верификации результатов тестирования сертифицируемых средств защиты информации.

При применении предложенного способа тестирования функций безопасности программно-аппаратных СЗИ в ФАУ «ГНИИИ ПТЗИ ФСТЭК России» стало возможным не только сократить временные издержки на трудозатратный процесс тестирования СЗИ, но и удостовериться в отсутствии негативного влияния сертифицируемых средств защиты на характеристики различных информационных систем.

Кроме того, результаты выполненных автором исследований были изучены и использованы в Национальном исследовательском ядерном университете (НИЯУ) «МИФИ» при разработке инновационных программно-аппаратных СЗИ в рамках исполнения работ по НИОКТР «Создание инженерно-технических решений для высокотехнологичного производства инновационных программно-аппаратных средств защиты информации на базе перспективных высокочастотных интерфейсов информационного взаимодействия».

В данной работе разрабатываются следующие средства защиты: программно-аппаратные комплексы контроля съемных носителей информации «Личный Секрет 3.0», «Секрет Фирмы 3.0», «Секрет Особого Назначения 3.0» и СОДС «МАРШ! 3.0». В процессе тестирования перечисленных ПАК контроля съемных носителей информации используются программы тестирования, разработанные автором в рамках данной диссертационной работы, а также адаптированный для применения с USB-3.0 вариант исполнения коммутатора USB-канала для автоматического подключения/отключения аппаратной компоненты к/от СВТ. Для тестирования «МАРШ!», в состав которого входит средство разграничения доступа «Аккорд-X», применяются программы тестирования «Аккорд-X» из разработанного комплекса программ, а также коммутатор USB-канала для подключения/отключения аппаратных идентификаторов. При этом проведенное в диссертации исследование и полученные результаты послужили теоретическим фундаментом и практическим руководством для успешного выполнения необходимых этапов работ по НИОКТР.

Также результаты диссертационного исследования используются в НИЯУ МИФИ в учебном процессе по дисциплине «Программно-аппаратные средства защиты информации». Предложенный способ тестирования программно-аппаратных СЗИ, алгоритмы их тестирования и верификации разбираются в лабораторных работах по данной дисциплине.

Реализация результатов диссертационной работы подтверждена соответствующими актами, описывающими практическое использование и внедрение результатов проведенных автором исследований. Акты внедрения приведены в Приложении Г.

4.3. Анализ опыта совместного использования разработанного программно-аппаратного комплекса для тестирования СЗИ и сторонних вспомогательных для тестирования средств

В разделе 4.1 проведены экспериментальные исследования предложенного способа и разработанных средств тестирования, показано, что проведение ручных проверок функций безопасности программно-аппаратных СЗИ с учетом достаточно быстрого изменения условий тестирования, например, выхода обновлений для ИС или ОС, приводит к неадекватным срокам внедрения этого средства защиты в ИС – для рассматриваемых в разделе СЗИ временные затраты на ручное тестирование составляют более полугода работы одного тестировщика. При этом было показано, что в случае применения предложенного способа на автоматическое тестирование и верификацию затрачивается не более одного-двух месяцев, что представляет собой вполне адекватные сроки. Однако большую часть времени, затрачиваемого на автоматическое тестирование занимает именно анализ результатов работы программ тестирования функций безопасности программно-аппаратных СЗИ. Это отчасти связано с большим объемом полученных результатов, но, в основном – со сложностью их анализа.

Помимо этого в процессе тестирования программно-аппаратных СЗИ существует необходимость проверки составляющих аппаратной компоненты, реализующих нецелевые функции, которые при этом взаимодействуют с функциями безопасности, реализуемые этой компонентой средства защиты. А это значит, что корректность работы таких нецелевых функций может оказывать влияние на работоспособность функций безопасности программно-аппаратного СЗИ.

Так для СЗИ, в состав которых входит флеш-память (а таких средств защиты в настоящее время большинство), необходимо проводить тестирование этой памяти, то есть: проверять надежность (на физическом уровне и на уровне абстракции файловой системы), определять скоростные характеристики операций чтения/записи, а также тестировать взаимодействие компонент, реализующих функции безопасности, с флеш-памятью. Необходимо отметить, что при тестировании флеш-памяти в аспекте определения ее надежности, целесообразно применять автоматическое или автоматизированное тестирование, так как ручное требует выполнения большого количества циклов атомарных операций чтения/записи различных по объему блоков данных и может затянуть внедрение СЗИ в ИС. При этом нет необходимости разрабатывать собственные программы тестирования, так как в открытом доступе существует ряд специализированных средств, позволяющих провести всестороннее тестирование флеш-памяти [34, 41, 42].

В связи с описанными аспектами целесообразным видится интеграция сторонних средств в процесс тестирования программно-аппаратных СЗИ в тех случаях, когда либо недостаточно применения используемых средств тестирования СЗИ (например, существует сложность анализа результатов работы программ тестирования), либо когда уже существуют сторонние специализированные средства для тестирования той или иной составляющей аппаратной компоненты СЗИ и нет смысла разрабатывать аналогичные им программы тестирования, когда можно использовать существующие (например, средства тестирования флеш-памяти для СЗИ с флеш-памятью).

Средства упрощения анализа результатов тестирования

При разработке любого продукта, в том числе программно-аппаратного СЗИ, возникает ряд сложностей, связанных с тем, что, как правило, тестировщик, составляющий ПМИ и тестировщик, разрабатывающий по нему программы тестирования (далее – автоматизатор), – это два разных человека. При этом за запуск программ тестирования отвечает не сам автоматизатор, а тестировщик, разработавший ПМИ. Также он должен проводить анализ результатов их выполнения, на что у него, не хватает ни знания самих программ, ни, как правило, квалификации [36]. В итоге это приводит к тому, что результаты выполнения программ тестирования может анализировать только автоматизатор, то есть [99]:

1. тестировщик не знает насколько программа тестирования соответствует ПМИ;
2. тестировщик не знает насколько программа тестирования покрывает ПМИ;
3. тестировщик не может анализировать отчеты с результатами выполнения программ тестирования, это должен делать автоматизатор.

Для преодоления данных сложностей можно использовать такие средства, как *Allure* [99], которые позволяют внести прозрачность в процесс создания и выполнения автоматических тестов, разработанных с применением уже используемых средств. Отчеты *Allure* помогают тестировщикам исключить перечисленные выше ситуации, а именно:

– для описания проверок используется предметно-ориентированный язык с последующим преобразованием в естественный язык при формировании отчета;

– к любому шагу программы тестирования можно прикладывать произвольное количество вложений (снимки экрана, HTML-страницы, заголовки запросов, ответы или серверные журналы

и так далее) и формировать подробное описание выполняемых проверок (так называемые Feature и Story), которые заносятся в отчет;

– возникновение ошибок сопровождается в отчете соответствующим статусом (ошибка в результате проверки – failed, возникшее исключение – broken, исключенные из запуска проверки – pending, проверки, пропущенные из-за ошибок в предусловии – assume failure).

При использовании таких сторонних средств как *Allure*, отдельно взятая программа тестирования становится понятным не только разработавшему ее автоматизатору, но и тестировщику, который запускает ее на выполнение – он может оценить соответствие и покрытие тестом проверок из ПМИ. Кроме того, в случае возникновения ошибки, в зависимости от статуса, тестировщик может определить что является причиной ошибки – тестируемая функция безопасности или сама программа тестирования.

Средства тестирования флеш-памяти

Как было рассмотрено выше, тестирование флеш-памяти необходимо выполнять наряду с тестированием функций безопасности программно-аппаратных СЗИ, так как некорректная работа данной памяти может оказывать влияние на работоспособность какой-либо функции безопасности. Необходимо отметить, что флеш-память некоторых СЗИ в процессе жизненного цикла может использоваться достаточно активно (такие случаи требуют особенно пристального внимания к характеристикам флеш-памяти). Например, флеш-память ПСКЗИ ШИПКА в составе ПАК «Центр-Т» используется достаточно часто: пользователь ежедневно загружает свой образ ПО ТС и тем самым задействует флеш-память [38, 76].

Также немаловажную роль при внедрении СЗИ в ИС играют скорости чтения/записи флеш-памяти, в зависимости от их значения, как правило, принимается решение о возможности использования данного средства защиты. Поэтому для СЗИ с флеш-памятью необходимо как гарантировать возможность качественной работы с ней, так и заранее определить, зафиксировать и опубликовать скорости чтения/записи при работе с устройством. Помимо этого разработчикам необходимо знать характеристики производимых устройств по параметрам, которые бы подтверждали корректность функционирования их флеш-памяти. Эти данные дадут возможность вовремя отбраковывать некорректно работающие устройства и устройства, с которыми в процессе эксплуатации могут возникнуть сложности [39].

Таким образом, существует два направления тестировании флеш-памяти, которые необходимо проводить для программно-аппаратных СЗИ: определение и подтверждение заявленных характеристик (скорости чтения/записи и других параметров) и нагрузочное тестирование [38].

В зависимости от типа флеш-памяти тестирование ее скорости состоит из измерения и анализа одного или нескольких следующих параметров [38]:

– время доступа при чтении и записи с использованием фиксированных тестовых блоков (4, 64 и 1024 Кбайт);

– скорость записи и чтения одного большого файла объемом несколько Гб;

– скорость записи и чтения множества маленьких файлов размером от нескольких байт до 500 Кбайт (например, сохраненные html-страницы, небольшие PDF-документы, материалы в форматах .doc и .xls и так далее).

Результатом такого тестирования является подтвержденная скорость чтения/записи флеш-памяти. Этот параметр производитель фиксирует и анонсирует впоследствии в качестве одной из характеристик разрабатываемого СЗИ с флеш-памятью, на основании которой можно судить о скорости работы с устройством.

Нагрузочные тесты подразделяются на:

– тестирование на физическом уровне (со всей доступной областью памяти). Результатом такого тестирования является показатель общей устойчивости флеш-памяти к частоте записи на нее произвольной информации (реальное количество возможных циклов перезаписи информации без нарушения нормального функционирования памяти). В данном случае проводится проверка именно равномерной записи данных на носитель;

– тестирование на уровне абстракции файловой системы (с разделами или логическими дисками). Результатом такого тестирования является показатель рабочей устойчивости памяти к частоте записи на нее произвольной информации, то есть учитывается запись в наиболее активные области раздела.

На основании полученных данных производитель может вовремя отбраковать некорректно работающие устройства, а также те устройства, с которыми в процессе эксплуатации могут возникнуть сложности.

Для проведения перечисленных проверок флеш-памяти программно-аппаратных СЗИ может использоваться ряд утилит, широкодоступных в настоящее время для тестирования устройств хранения информации (HDD, CD-ROM, Flash, floppy и так далее), например, утилиты *HD_Speed* и *CheckFlash* [38, 72, 93].

Утилита *HD_Speed* предназначена для тестирования скорости чтения/записи устройств хранения информации. *HD_Speed* позволяет задать смещение и размер считываемого/записываемого блока данных, выбор этих параметров может повлиять на скорость работы флеш-памяти. Помимо этого можно выбрать отдельный логический раздел и проверить, как, например, конкретная файловая система или наличие «плохих» секторов (англ. bad sectors, bad blocks) влияет на производительность флеш-памяти.

Check Flash – утилита, предназначенная для проверки надежности флеш-памяти, также позволяющая определить мгновенную и среднюю скорость чтения/записи (для блоков данных различного размера). *Check Flash* позволяет проверить стабильность чтения/записи (как на логическом, так и на физическом уровне) с заданным количеством циклов или до возникновения первой ошибки.

При использовании обеих утилит можно определить количество ошибок чтения/записи, возникших в процессе тестирования, а вся полученная информация сохраняется в журнал для дальнейшего анализа. На основании полученных результатов можно сделать вывод о пригодности флеш-памяти для использования в составе программно-аппаратного СЗИ.

Интеграция рассмотренных средств позволит упросить процесс тестирования таких средств защиты и еще больше сократить временные затраты на его проведение совместно с предложенными в диссертационной работе решениями, а значит, уменьшить сроки внедрения средства защиты в ИС.

4.4. Выводы

Путем экспериментальных исследований в главе получены следующие результаты:

1. Проведены апробация и анализ результатов применения разработанных способа тестирования функций безопасности и программно-аппаратного комплекса «Тестирование функций безопасности программно-аппаратных СЗИ», а также проведена его апробация, результаты которой подтвердили их работоспособность и состоятельность результатов диссертационной работы как на качественном, так и на количественном уровне.

2. Подтверждено, что при применении предложенного способа и разработанных средств тестирования для функций безопасности программно-аппаратных СЗИ становится возможным обеспечить полноту и оптимальность проводимого тестирования.

3. Продемонстрирована целесообразность использования автоматического тестирования и верификации для функций безопасности программно-аппаратных СЗИ. Тем самым решается задача принципиальной возможности тестирования функций безопасности программно-аппаратных СЗИ в требуемые сроки.

4. Показана целесообразность интеграции в процесс тестирования программно-аппаратных СЗИ сторонних средств, облегчающих анализ результатов работы программ тестирования и делающих процесс автоматического тестирования более прозрачным, а также средств тестирования флеш-памяти при наличии таковой в составе программно-аппаратного средства защиты информации. Такая интеграция совместно с предложенными в диссертационной работе решениями позволит упростить процесс тестирования таких СЗИ и еще больше сократить временные затраты на его проведение, а значит, уменьшить сроки их внедрения в ИС.

Содержание диссертации и основные положения, выносимые на защиту, отражают персональный вклад автора в работу. Все основные представленные в диссертации результаты получены автором самостоятельно.

ЗАКЛЮЧЕНИЕ

В диссертационной работе проведено моделирование состояний программно-аппаратных средств защиты информации и предложен научно-обоснованный способ тестированияих функций безопасности, учитывающий состояния аппаратной компоненты и применимый для различных видов программно-аппаратных СЗИ. Данный способ формирует порядок использования предложенных критериев применимости существующих способов тестирования ПО, алгоритмов тестирования и верификации для функций безопасности таких средств защиты информации.

В результате проведенного исследования были получены следующие основные результаты:

1. Проведенный анализ существующих способов и средств тестирования ПО применительно к программно-аппаратным СЗИ выявил актуальную задачу моделирования их состояний для разработки нового научно-обоснованного способа тестирования таких средств защиты, а также средств их тестирования, позволяющих автоматизировать данный процесс.

2. Предложеная модель программно-аппаратных СЗИ, учитывающая состояния аппаратной компоненты, позволяет представить любое программно-аппаратное средство защиты в виде конечного детерминированного автомата – совокупности множеств состояний программной и аппаратной компоненты СЗИ; стимулов, в том числе реализуемых и подлежащих тестированию функций безопасности; реакций и переходов автомата из одного состояния в другое.

3. Предложенные критерии применимости существующих способов тестирования ПО к различным видам функций безопасности программно-аппаратных СЗИ, устанавливающие формальные правила определения возможности выполнения проверок и учитывающие особенности всевозможных видов функций безопасности, их зависимости от переходов из состояние в состояние, позволяют для конкретного средства защиты: определить принципиальную возможность применения, а также необходимость разработки нового научно-обоснованного способа и средств тестирования; выявить функции безопасности и переходы, которые препятствуют применению существующих способов тестирования ПО; выделить перечень проверок функций безопасности, выполнимых в зависимости от текущего состояния СЗИ; обосновать необходимость применения вспомогательных средств для обеспечения выполнимости всех проверок функций безопасности.

4. Разработанный алгоритм тестирования функций безопасности программно-аппаратных СЗИ, основанный на известных положениях теории графов, позволяет получить решение задачи тестирования, а также обеспечить его полноту и оптимальность.

5. Разработанный алгоритм верификации функций безопасности программно-аппаратных СЗИ, основанный на известных положениях теории оптимизации и принятия решений и содержащий процедуры оценки критичности выявленных в ходе тестирования ошибок, принимающий критичность каждой ошибки в виде совокупности уровней критичности самой ошибки и функции безопасности, в которой она обнаружена, позволяет оценить степень влияния таких ошибок на защищенность системы, на основании чего принять решение об успешном завершении тестирования или о возврате СЗИ на доработку.

6. Предложенный способ тестирования функций безопасности программно-аппаратных СЗИ, учитывающий возможные состояния аппаратной компоненты и устанавливающий порядок использования разработанных критериев применимости существующих способов тестирования ПО для таких средств защиты, алгоритмов тестирования и верификации их функций безопасности, позволяет проводить тестирование различных видов функций безопасности программно-аппаратных СЗИ и обеспечить полноту и оптимальность данного тестирования.

7. Предложенные рекомендации и разработанный на их основе алгоритм тестирования функций безопасности программно-аппаратных СЗИ с применением средств виртуализации, позволяют обеспечить возможность выполнения некоторых функций безопасности в виртуальных машинах при невозможности их выполнения на реальных аппаратных платформах.

8. Предложенные рекомендации по практической реализации вспомогательных средств тестирования позволили реализовать коммутатор USB-канала, полностью эмулирующий физическое отключение и подключение СЗИ к СВТ как на уровне питания, так и на уровне канала передачи данных. Данный коммутатор позволяет выполнять отключение и подключение СЗИ с USB-интерфейсом к/от СВТ в программах тестирования и тем самым автоматизировать тестирование их функций безопасности.

9. Предложенные рекомендации по практической реализации программ тестирования и программы верификации для различных видов программно-аппаратных СЗИ позволили реализовать программный комплекс «Тестирование функций безопасности программно-аппаратных СЗИ». Данный комплекс совместно с разработанным коммутатором USB-канала позволяет выполнять автоматическое тестирование и верификацию различных видов программно-аппаратных средств защиты информации, и тем самым обеспечить принципиальную возможность тестирования их функций безопасности в требуемые сроки.

10. Проведенные апробация и анализ результатов применения разработанного способа тестирования функций безопасности и основанного на нем программно-аппаратного комплекса для тестирования СЗИ подтвердили качественное и количественное улучшение результатов тестирования функций безопасности программно-аппаратных СЗИ при их применении. При использовании результатов работы становится возможным обеспечить полноту и оптимальность тестирования, при котором проверка каждой функции безопасности проводится во всех возможных состояниях, а также понизить сложность и временные затраты на его проведение, уменьшить количество непроводимых ранее проверок функций безопасности и увеличить общее число проверок путем учета всех возможных переходов.

11. Основные результаты диссертационной работы, касающиеся разработки модели, алгоритмов тестирования и верификации, способа и средств тестирования программно-аппаратных СЗИ, используются в рабочем процессе в ЗАО «ОКБ САПР» и ФАУ «ГНИИИ ПТЗИ ФСТЭК России», в НИОКТР в НИЯУ МИФИ. Аналитические результаты применены в учебном процессе кафедры «Криптология и кибербезопасность» НИЯУ МИФИ в рамках дисциплины «Программно-аппаратные средства защиты информации». Имеются четыре акта о внедрении результатов диссертационной работы.

В заключении хотелось бы отметить, что в качестве дальнейших перспектив развития темы диссертационного исследования целесообразным видится усовершенствование разработанного способа и алгоритмов, доработка предложенного вспомогательного средства тестирования для функций безопасности программно-аппаратных СЗИ других видов или имеющих другой интерфейс подключения, а также для предоставления возможности поочередного подключения тестируемых средств защиты в несколько различных СВТ. Помимо этого, возможна адаптация входящих в разработанный комплекс программ тестирования для функций безопасности других СЗИ.

Список сокращений и условных обозначений

BSOD	Синий экран смерти (англ. Blue Screen of Death)
SDK	Software Development Kit
АМДЗ	Аппаратный модуль доверенной загрузки
АРМ	Автоматизированное рабочее место
АСУ ТП	Автоматизированая система управления производственными и технологическими процессами
ВМ	Виртуальная машина
ГИС	Государственная информационная система
ДСС	Доверенный сеанс связи
ИС	Информационная система
ИСПДн	Информционные системы персональных данных
КА	Коды аутентификации
КИИ	Критическая информационная инфраструктура
КЦ	Контроль целостности
НСД	Несанкционированный доступ
ОС	Операционная система
ПАК	Программно-аппаратный комплекс
ПМИ	Программа и методика испытаний
ПО	Программное обеспечение
СВТ	Средство вычислительной техники
СЗИ	Средство защиты информации
СОДС	Средство обеспечения доверенного сеанса
СХСЗ	Сервер хранения и сетевой загрузки
ТС	Терминальная станция
ТТ	Технические требования

Список литературы

1. Автоматизация тестирования программных систем. [Электронный ресурс] – URL: https://habr.com/ru/post/160257/ (дата обращения: 25 апреля 2023 г.).
2. Алгоритмы: построение и анализ / Т. Кормен [и др.]. — 3-е изд. — М. : Вильямс, 2013. — 1328 с.
3. *Александров, П. С.* Введение в теорию множеств и общую топологию: Учебное пособие. 2 изд. / П. С. Александров. — Спб. : Изд-во «Лань», 2010. — Т. 1. — 368 с.
4. *Бажитов, И. А.* Возможности ПАК СЗИ НСД «АККОРД-Х» для ОС Linux / И. А. Бажитов // *Комплексная защита информации. Сборник материалов XIV международной конференции 19-22 мая 2009 г., Минск (Республика Беларусь)*. — 2009. — С. 26–27.
5. *Бажитов, И. А.* Построение автономной защищенной системы загрузки и хранения ПО терминальных станций / И. А. Бажитов // *Комплексная защита информации. Сборник материалов XV международной конференции 1-4 июня 2010 г., Иркутск (Россия)*. — 2010. — 35 с.
6. *Бейзер, Б.* Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем / Б. Бейзер. — СПб. : Питер, 2004. — 320 с.
7. *Блэк, Р.* Ключевые процессы тестирования / Р. Блэк. — М. : Лори, 2006. — 566 с.
8. *Братищенко, В. В.* Проектирование информационных систем / В. В. Братищенко. — Иркутск : Изд-во БГУЭП, 2004. — 84 с.
9. *Брауде, Э.* Технология разработки программного обеспечения / Э. Брауде. — СПб. : Питер, 2004. — 655 с.
10. *Бурдонов, И. Б.* Использование конечных автоматов для тестирования программ / И. Б. Бурдонов, А. С. Косачев, В. В. Кулямин // *Программирование*. — 2000. — № 2. — С. 12–28.
11. *Бэк, К.* Экстремальное программирование: разработка через тестирование / К. Бэк. — СПб. : Питер, 2003. — 224 с.
12. *Веретенников, А.* Классификация средств защиты информации от ФСТЭК и ФСБ России. [Электронный ресурс] – URL: https://www.anti-malware.ru/analytics/Market_Analysis/infosecurity-systems-classification-fsb-fstek (дата обращения: 25 апреля 2023 г.).
13. *Винниченко, И. В.* Автоматизация процессов тестирования / И. В. Винниченко. — СПб. : Питер, 2005. — 203 с.
14. *Вонг, А.* Оптимизация BIOS. Полный справочник по всем параметрам BIOS и их настройкам / А. Вонг. — М. : ДМК Пресс, 2011. — 272 с.
15. ГОСТ 28195-89. Оценка качества программных средств. Общие положения. — М. : Стандартинформ, 1990. — 30 с.
16. ГОСТ Р 19.301-79. Программа и методика испытаний. Требования к содержанию и оформлению. — М. : Стандартинформ, 2010. — 2 с.
17. ГОСТ Р 50922-2006. Защита информации. Основные термины и определения. — М. : Стандартинформ, 2008. — 8 с.

18. ГОСТ Р 51901.12-2007. Менеджмент риска. Метод анализа видов и последствий отказов. — М. : Стандартинформ, 2008. — 36 с.
19. *Грекул, В. И.* Проектирование информационных систем / В. И. Грекул, Г. Н. Денищенко, Н. Л. Коровкина. — М. : Интернет-университет информационных технологий, 2005. — 304 с.
20. *Гук, М. Ю.* Аппаратные интерфейсы ПК / М. Ю. Гук. — СПб. : Питер, 2002. — 528 с.
21. *Гэртнер, М.* ATDD. Разработка программного обеспечения через приемочные тесты / М. Гэртнер. — М. : ДМК Пресс, 2013. — 232 с.
22. *Дастин, Э.* Автоматизированное тестирование программного обеспечения. Внедрение, управление и эксплуатация / Э. Дастин, Дж. Рэшка, Дж. Пол. — М. : Лори, 2003. — 592 с.
23. *Долгова, К. Н.* О некоторых задачах обратной инженерии / К. Н. Долгова, А. В. Чернов // *Учреждение Российской академии наук. Институт системного программирования РАН*. — 2008. — Т. 15. — С. 119–134.
24. *Дроботун, Е. Б.* Критичность ошибок в программном обеспечении и анализ их последствий / Е. Б. Дроботун // *Фундаментальные исследования*. — 2009. — № 4. — С. 73–74.
25. *Журавлев, Ю. И.* Дискретный анализ. Формальные системы и алгоритмы / Ю. И. Журавлев, Ю. А. Флеров, М. Н. Вялый. — М. : Контакт Плюс, 2010. — 336 с.
26. *Калбертсон, Р.* Быстрое тестирование / Р. Калбертсон, К. Браун, Г. Кобб. — М. : Вильямс, 2002. — 384 с.
27. *Канер, С.* Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес приложений / С. Канер. — Киев : ДиаСофт, 2001. — 544 с.
28. *Каннер, Т. М.* Особенности применения средств виртуализации при тестировании программно-аппаратных средств защиты информации / Т. М. Каннер // *Информация и безопасность*. — 2015. — Т. 18, № 3. — С. 416–419.
29. *Каннер, Т. М.* Применимость методов тестирования ПО к программно-аппаратным СЗИ / Т. М. Каннер // *Вопросы защиты информации*. — 2015. — № 1. — С. 30–39.
30. *Каннер, Т. М.* Коммутатор USB-канала как техническое средство для тестирования программно-аппаратных СЗИ / Т. М. Каннер // *Комплексная защита информации. Материалы XXI Международной конференции 17-19 мая 2016 года, Смоленск (Россия)*. — 2016. — С. 200–204.
31. *Каннер, Т. М.* Разработка и оценка эффективности применения средств тестирования функций безопасности программно-аппаратных СЗИ / Т. М. Каннер // *Информация и безопасность*. — 2017. — № 3. — С. 330–333.
32. *Каннер, Т. М.* Эффективность применения средств тестирования программно-аппаратных СЗИ / Т. М. Каннер // *Вопросы защиты информации*. — 2017. — № 2. — С. 9–13.
33. *Каннер, Т. М.* Адаптация существующих способов верификации для программно-аппаратных СЗИ / Т. М. Каннер // *Вопросы защиты информации*. — 2018. — № 1. — С. 13–19.
34. *Каннер, Т. М.* Формирование подхода к автоматизации тестирования СЗИ, в конструктив которых входит флеш-память, функционирующих в ОС / Т. М. Каннер, К. А. Куваева // *Вопросы защиты информации*. — 2014. — № 4. — С. 52–54.
35. *Каннер, Т. М.* О выборе инструмента автоматизации тестирования для программно-

аппаратных СЗИ / Т. М. Каннер, А. И. Обломова // *Вопросы защиты информации*. — 2014. — № 4. — С. 34–36.

36. *Каннер, Т. М.* Особенности верификации средств защиты информации / Т. М. Каннер, Х. С. Султанахмедов // *Вопросы защиты информации*. — 2014. — № 4. — С. 55–57.
37. *Каннер(Борисова), Т. М.* Особенности автоматизации тестирования программно-аппаратных СЗИ / Т. М. Каннер(Борисова) // *Безопасность информационных технологий*. — 2013. — № 2. — С. 27–31.
38. *Каннер(Борисова), Т. М.* Особенность тестирования СЗИ, в конструктив которых входит флеш-память / Т. М. Каннер(Борисова) // *Комплексная защита информации. Материалы XVIII Международной конференции 21-24 мая 2013 года, Брест (Республика Беларусь)*. — 2013. — № 6. — С. 119–120.
39. *Каннер(Борисова), Т. М.* Задача тестирования аппаратных средств защиты информации / Т. М. Каннер(Борисова), В. А. Гадасин // *Вопросы защиты информации*. — 2012. — № 3. — С. 10–16.
40. *Каннер(Борисова), Т. М.* Проблемы тестирования СЗИ, функционирующих до старта ОС / Т. М. Каннер(Борисова), А. В. Кузнецов // *Комплексная защита информации. Материалы XVIII Международной конференции 21-24 мая 2013 года, Брест (Республика Беларусь)*. — 2013. — № 6. — С. 114–115.
41. *Каннер(Борисова), Т. М.* Тестирование средств защиты информации / Т. М. Каннер(Борисова), А. В. Кузнецов, А. И. Обломова // *Информационная безопасность. Материалы XIII Международной конференции 9-12 июля 2013 года, Таганрог (Россия)*. — 2013. — Т. 1. — С. 121–129.
42. *Каннер(Борисова), Т. М.* Способы автоматизации тестирования СЗИ, функционирующих в ОС, на примере ПСКЗИ ШИПКА / Т. М. Каннер(Борисова), А. И. Обломова // *Комплексная защита информации. Материалы XVIII Международной конференции 21-24 мая 2013 года, Брест (Республика Беларусь)*. — 2013. — № 6. — С. 117–118.
43. *Конявская, С. В.* Идеальное устройство. ПСКЗИ ШИПКА / С. В. Конявская // *Connect! Мир связи*. — 2008. — № 10. — 111 с.
44. *Конявская, С. В.* Секреты Особого Назначения. Доказуемость благонадежности / С. В. Конявская // *Информационные технологии, связь и защита информации МВД России. Ч. 2*. — 2012. — 238 с.
45. *Конявская, С. В.* Аппаратные СЗИ НСД. Повторение пройденного / С. В. Конявская, М. Г. Мищенко, С. А. Синякин // *Управление защитой информации*. — 2007. — Т. 11 №1. — С. 53–56.
46. *Конявская, С. В.* Аппаратная криптография. Особенности «тонкой» настройки / С. В. Конявская, Д. Ю. Счастный, Т. М. Каннер // *INSIDE*. — 2010. — № 5. — С. 40–44.
47. *Конявский, В. А.* Доверенный сеанс связи. Развитие парадигмы доверенных вычислительных систем – на старт, внимание, МАРШ! / В. А. Конявский // *Комплексная защита информации. Сборник материалов XV международной конференции 1-4 июня 2010 г., Иркутск (Россия)*. — 2010. — С. 166–169.
48. *Конявский, В. А.* Компьютерная преступность / В. А. Конявский, С. В. Лопаткин. — М. : РФК-Имидж Лаб, 2006. — Т. 1. — 560 с.

49. *Котляров, В. П.* Основы тестирования программного обеспечения / В. П. Котляров, Т. В. Коликова. — М. : Бином, 2006. — 285 с.
50. *Криспин, Л.* Гибкое тестирование. Практическое руководство для тестировщиков ПО и гибких команд / Л. Криспин. — М. : Вильямс, 2010. — 464 с.
51. *Кулямин, В. В.* Тестирование на основе моделей. Курс лекции ВМиК МГУ. [Электронный ресурс] – URL: http://panda.ispras.ru/~kuliamin/mbt-course.html (дата обращения: 25 апреля 2023 г.).
52. *Кулямин, В. В.* Методы верификации программного обеспечения / В. В. Кулямин. — М. : Институт системного программирования РАН, 2008. — 111 с.
53. *Липаев, В. В.* Качество программного обеспечения / В. В. Липаев. — М. : Финансы и статистика, 1983. — 264 с.
54. *Липаев, В. В.* Тестирование программ / В. В. Липаев. — М. : Радио и связь, 1986. — 296 с.
55. *Лыдин, С. С.* Подходы к построению системы защищенного применения служебных носителей информации / С. С. Лыдин // *Комплексная защита информации. Безопасность информационных технологий. Материалы XVII Международной конференции 15-18 мая 2012 г., Суздаль (Россия)*. — 2012. — С. 167–169.
56. МАРШ! [Электронный ресурс] – URL: https://www.okbsapr.ru/products/storage/compute/sods/marsh/ (дата обращения: 25 апреля 2023 г.).
57. *Майерс, Г.* Надежность программного обеспечения / Г. Майерс. — М. : Мир, 1980. — 359 с.
58. *Майерс, Г.* Искусство тестирования программ / Г. Майерс. — М. : Финансы и статистика, 1982. — 176 с.
59. *Майерс, Г.* Искусство тестирования программ / Г. Майерс, Т. Баджетт, К. Сандлер. — 3-е изд. — М. : Вильямс, 2012. — 271 с.
60. *Макгрегор, Д.* Тестирование объектно-ориентированного программного обеспечения / Д. Макгрегор, Д. Сайкс. — М. : ДиаСофт, 2002. — 412 с.
61. *Макейчик, Ю. С.* Аккорд-NT/2000 V.3.0 REVISION 4. Что нового / Ю. С. Макейчик // *Комплексная защита информации. Сборник материалов XII Международной конференции 13-16 мая 2008 г., Ярославль (Россия)*. — 2008. — С. 134–135.
62. *Макконнелл, С.* Совершенный код. Мастер-класс / С. Макконнелл. — М. : Издательско-торговый дом «Русская редакция», 2005. — 896 с.
63. *Миноженко, А. В.* Способ поиска уязвимостей ПО путем изменения входящих параметров с использованием статических данных / А. В. Миноженко // *Аннотированный сборник научно-исследовательских выпускных квалификационных работ студентов НИУ ИТМО*. — 2011. — С. 23–38.
64. Мобильное устройство защиты информации : патент на полезную модель № 83862 Российская Федерация / Т. М. Каннер(Борисова) [и др.]. — опубл. 20.06.2009, бюл. № 17.
65. *Орлик, С.* Введение в программную инженерию и управление жизненным циклом ПО. Программная инженерия / С. Орлик, Ю. Булуй. — М. , 2005. — 183 с.
66. Персональное средство криптографической защиты информации ШИПКА. Руководство администратора. 11443195.4012-022 90 01 // *ОКБ САПР*. — 2012. — 124 с.

67. *Петров, С. В.* Шины PCI, PCI Express. Архитектура, дизайн, принципы функционирования / С. В. Петров. — СПб. : БХВ-Петербург, 2006. — 416 с.
68. *Пивень, А. А.* Тестирование программного обеспечения / А. А. Пивень, Ю. И. Скорин // *Системы обработки информации*. — 2012. — Т. 1, № 4. — С. 56–58.
69. *Плаксин, М. А.* Тестирование и отладка программ – для профессионалов будущих и настоящих / М. А. Плаксин. — М. : Бином. Лаборатория знаний, 2007. — 169 с.
70. Программно-аппаратный комплекс СЗИ от НСД для ПЭВМ (РС) «Аккорд-АМДЗ». Руководство администратора. 11443195.4012-038 90 2011 // *ОКБ САПР*. — 2014. — 54 с.
71. Программно-аппаратный комплекс Соболь Версия 3.0. Руководство администратора. RU.40308570.501410.001 91 1 // *Код Безопасности*. — 2016. — 39 с.
72. Программы для тестирования скорости USB-накопителей. [Электронный ресурс] – URL: http://www.spec-service.com/Pages/109.htm (дата обращения: 25 апреля 2023 г.).
73. *Рассел, Д.* Жизненный цикл программного обеспечения / Д. Рассел. — М. : Книга по Требованию, 2012. — 89 с.
74. *Роббинс, Дж.* Отладка приложений для Microsoft. NET и Microsoft Windows / Дж. Роббинс. — СПб. : Русская Редакция, Питер, 2004. — 512 с.
75. *Ройс., У.* Управление проектами по созданию программного обеспечения / У Ройс. — М. : Лори, 2002. — 448 с.
76. Руководство Администратора АРМ «Центр». 11443195.4012.042 90 // *ОКБ САПР*. — 2014. — 45 с.
77. *Савин, Р.* Тестирование Дот Ком, или Пособие по жестокому обращению с багами в интернет-стартапах / Р. Савин. — М. : Дело, 2007. — 312 с.
78. *Савицкий, В. О.* Инкрементальный анализ исходного кода на языках C/C++ / В. О. Савицкий // *Учреждение Российской академии наук. Институт системного программирования РАН*. — 2012. — Т. 22. — С. 119–129.
79. *Саттон, М.* Fuzzing: исследование уязвимостей методом грубой силы / М. Саттон, А. Грин, П. Амини. — Спб. : Символ-Полюс, 2009. — 560 с.
80. Свидетельство о государственной регистрации программы для ЭВМ. Заявка 2016616332. Российская Федерация. Тестирование функций безопасности программно-аппаратных средств защиты информации / Авторы Каннер Т. М., Коробов В. В., правообладатель ЗАО «ОКБ САПР». — № 2016616332; заявл. 14.04.2016; опубл. 09.06.2016 (приравнивается к публикации в журнале из Перечня ВАК).
81. *Седжвик, Р.* Фундаментальные алгоритмы на С. Часть 5: Алгоритмы на графах / Р. Седжвик. — 3-е изд. — СПб. : ДиаСофтЮП, 2003. — 496 с.
82. *Седжвик, Р.* Алгоритмы на Java / Р. Седжвик, К. Уэйн. — 4-е изд. — М. : Вильямс, 2016. — 848 с.
83. *Синицын, С. В.* Верификация программного обеспечения / С. В. Синицын, Н. Ю. Налютин. — М. : Бином, 2008. — Т. 6. — 368 с.
84. *Синякин, С. А.* Особенности совместимости Аккорд-АМДЗ и современных СВТ / С. А. Синякин // *Комплексная защита информации. Электроника инфо. Материалы XVIII Междуна-*

родной конференции 21-24 мая 2013 года, Брест (Республика Беларусь). — 2013. — № 6. — С. 142–144.

85. Состав и содержание организационных и технических мер по обеспечению безопасности персональных данных при их обработке в информационных системах персональных данных. Утвержден приказом ФСТЭК России от 18 февраля 2013 г. №21. [Электронный ресурс] – URL: https://fstec.ru/normotvorcheskaya/akty/53-prikazy/691-prikaz-fstek-rossii-ot-18-fevralya-2013-g-n-21 (дата обращения: 25 апреля 2023 г.).
86. *Степанченко, И. В.* Методы тестирования программного обеспечения / И. В. Степанченко. — Волгоград : ВолгГТУ, 2006. — 76 с.
87. *Столлингс, В.* Структурная организация и архитектура компьютерных систем / В. Столлингс. — 5-е изд. — М. : Вильямс, 2002. — 896 с.
88. *Тамре, Л.* Введение в тестирование программного обеспечения / Л. Тамре. — М. : Вильямс, 2003. — 354 с.
89. Теоретические основы компьютерной безопасности: Учебн. пособие для вузов. / П. Н. Девянин [и др.]. — М. : Радио и связь, 2000. — 192 с.
90. Требования к обеспечению защиты информации в автоматизированных системах управления производственными и технологическими процессами на критически важных объектах, потенциально опасных объектах, а также объектах, представляющих повышенную опасность для жизни и здоровья людей и окружающей среды. Утверждены приказом ФСТЭК России от 14 марта 2014 г. №31. [Электронный ресурс] – URL: https://fstec.ru/normotvorcheskaya/akty/53-prikazy/868-prikaz-fstek-rossii-ot-14-marta-2014-g-n-31 (дата обращения: 25 апреля 2023 г.).
91. Требования о защите информации, не составляющей государственную тайну, содержащейся в государственных информационных системах. Утверждены приказом ФСТЭК России от 11 февраля 2013 г. №17. [Электронный ресурс] – URL: https://fstec.ru/normotvorcheskaya/akty/53-prikazy/702-prikaz-fstek-rossii-ot-11-fevralya-2013-g-n-17 (дата обращения: 25 апреля 2023 г.).
92. Требования по обеспечению безопасности значимых объектов критической информационной инфраструктуры Российской Федерации. Утверждены приказом ФСТЭК России от 25 декабря 2017 г. № 239. [Электронный ресурс] – URL: https://fstec.ru/normotvorcheskaya/akty/53-prikazy/1592-prikaz-fstek-rossii-ot-25-dekabrya-2017-g-n-239 (дата обращения: 25 апреля 2023 г.).
93. Утилиты для тестирования скорости флэшек. [Электронный ресурс] – URL: http://old.computerra.ru/reviews/415531/ (дата обращения: 25 апреля 2023 г.).
94. *Черноруцкий, И. Г.* Методы оптимизации и принятия решений / И. Г. Черноруцкий. — СПб. : Лань, 2001. — 381 с.

95. *Чуканов, В. О.* Надежность программного обеспечения и аппаратных средств систем передачи данных атомных электростанций / В. О. Чуканов. — М. : МИФИ, 2008. — 166 с.
96. *Язов, Ю. К.* Перспективы развития систем поддержки принятия решений в области защиты информации / Ю. К. Язов, С. В. Соловьев, Е. В. Колесникова // *Комплексная защита информации. Материалы XXI Международной конференции 17-19 мая 2016 года, Смоленск (Россия)*. — 2016. — С. 78–81.
97. Parallels Desktop User's Guide. Работа в режиме Coherence. [Электронный ресурс] – URL: http://download.parallels.com/desktop/v4/docs/ru/Parallels_Desktop_Users_Guide/23413.htm (дата обращения: 25 апреля 2023 г.).
98. AMD I/O Virtualization Technology (IOMMU) Specification Revision 1.26. [Электронный ресурс] – URL: http://developer.amd.com/wordpress/media/2012/10/34434-IOMMU-Rev_1.26_2-11-09.pdf (дата обращения: 25 апреля 2023 г.).
99. Allure: Test report and framework for writing self-documented tests. [Электронный ресурс] – URL: http://allure.qatools.ru/ (дата обращения: 25 апреля 2023 г.).
100. *Alpaev, G.* TestComplete Cookbook / G. Alpaev. — UK : Packt Publishing, 2013. — 282 p.
101. *Edmonds, J.* Matching Euler tours and the Chinese postman / J. Edmonds, E. L. Johnson // *Mathematical programming*. — 1973. — Vol. 5, no. 1. — Pp. 88–124.
102. Intel Virtualization Technology for Directed I/O (VT-d) Architecture Specification. [Электронный ресурс] – URL: http://www.intel.com/content/www/us/en/embedded/technology/virtualization/vt-directed-io-spec.html (дата обращения: 25 апреля 2023 г.).
103. Intel®64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide. — 2015.
104. *Kanner, T. M.* Applicability of Software Testing Methods to Software and Hardware Data Security Tools / T. M. Kanner // *Global Journal of Pure and Applied Mathematics*. — 2016. — Vol. 12, no. 1. — Pp. 167–190.
105. *Kanner, T. M.* Applying a Mathematical Approach to Interpreting the Results of Testing Software and Hardware Data Security Tools during the Verification Process / T. M. Kanner, A. M. Kanner // *Journal of Engineering and Applied Sciences*. — 2019. — Vol. 14, no. 10. — Pp. 3482–3491.
106. *Kanner, T. M.* Testing Software and Hardware Data Security Tools Using the Automata Theory and the Graph Theory / T. M. Kanner, A. M. Kanner // *2020 Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology (USBEREIT 2020)*. — 2020. — Pp. 615–618.
107. *Kanner, T. M.* Comprehensive Testing of Software and Hardware Data Security Tools Using Virtualization / T. M. Kanner, A. M. Kanner, A.V. Epishkina // *Advanced Technologies in Robotics and Intelligent Systems. Mechanisms and Machine Science*. — 2020. — Vol. 80. — Pp. 79–87.
108. *Kanner, T. M.* Algorithm for Optimal and Complete Testing of Software and Hardware Data Security Tools / T. M. Kanner, A. M. Kanner, A.V. Epishkina // *Procedia Computer Science*. — 2021. — Vol. 190. — Pp. 408–413.
109. *Lau, B.* Instant Sikuli Test Automation / B. Lau. — UK : Packt Publishing, 2013. — 54 p.

110. Oracle VM VirtualBox User Manual. Seamless windows. [Электронный ресурс] – URL: https://www.virtualbox.org/manual/ch04.html#seamlesswindows (дата обращения: 25 апреля 2023 г.).
111. *Sharir, M.* A strong connectivity algorithm and its applications to data flow analysis / M. Sharir // *Computers and Mathematics with Applications*. — 1981. — Vol. 7, no. 1. — Pp. 67–72.
112. *Skiena, S. S.* The Algorithm design manual / S. S. Skiena. — 2nd edition. — London : Springer, 2010. — 748 p.
113. VMware Workstation Documentation Center. Use Unity Mode. [Электронный ресурс] – URL: https://pubs.vmware.com/workstation-9/index.jsp (дата обращения: 25 апреля 2023 г.).
114. *Zimmer, V.* Harnessing the UEFI Shell: Moving the Platform Beyond / V. Zimmer, T. Lewis, M. Rothman. — Intel Press, 2010. — 413 p.

Список иллюстративного материала

Список рисунков

Список таблиц

Приложение А

Перечень проверок для тестирования функций безопасности и нецелевых функций мобильных программно-аппаратных СЗИ

Функций проверок для тестирования функций безопасности и нецелевых функций мобильных программно-аппаратных СЗИ:

– *function findKeyItem (keyList, keyAlgoritm, keyLength, keyExport, keyType)* – получение строки с заданными параметрами ключа (*keyList* – список ключей, *keyAlgoritm* – алгоритм ключа, *keyLength* – длина ключа, *keyExport* – экспортируемость ключа, *keyType* – тип ключа, симметричный/открытый/закрытый; возвращает −1, если строка отсутствует в списке ключей);

– *function findKeyItemEncSing(keyList, keyAlgoritm, keyLength, keyExport)* – получение строки с заданными параметрами ключа для списка ключей шифрования (*keyList* – список ключей, *keyAlgoritm* – алгоритм ключа, *keyLength* – длина ключа, *keyExport* – экспортируемость ключа; возвращает −1, если строка отсутствует в списке ключей);

– *function getCountKeys(keyList, keyAlgoritm, keyLength, keyExport, keyType)* – получение количества найденных ключей в списке по заданным параметрам (*keyList* – список ключей, *keyAlgoritm* – алгоритм ключа, *keyLength* – длина ключа, *keyExport* – экспортируемость ключа, *keyType* – тип ключа, симметричный/открытый/закрытый);

– *function checkKeyParam(wGenKeyForm, keyAlgoritm, keyLength, keyExport)* – проверка корректности параметров ключа (*wGenKeyForm* – «ссылка» на форму генерации ключей, *keyAlgoritm* – алгоритм ключа, *keyLength* – длина ключа, *keyExport* – экспортируемость ключа);

– *function genSymKey(wGenKey, keyAlgoritm, keyLength, keyExport)* – генерация симметричного ключа: проверка параметров и факта создания ключа (*wGenKeyForm* – «ссылка» на форму генерации ключей, *keyAlgoritm* – алгоритм ключа, *keyLength* – длина ключа, *keyExport* – экспортируемость ключа);

– *function genAsyKey(wGenKey, keyAlgoritm, keyLength, keyExport)* – генерация ключевых пар: проверка параметров и факта создания ключей ключевой пары (*wGenKey* – форма генерации ключей, *keyAlgoritm* – алгоритм ключа, *keyLength* – длина ключа, *keyExport* – экспортируемость ключа;

– *function exportSymKey (exKeyAlgoritm, exKeyLength, exKeyExport, opKeyAlgoritm)* – экспорт симметричного ключа и проверка создания файла с экспортированным ключом (*exKeyAlgoritm* – алгоритм экспортируемого ключа, *exKeyLength* – длина экспортируемого ключа, *exKeyExport* – возможность экспорта ключа, *opKeyAlgoritm* – алгоритм открытого ключа, на котором выполняется экспорт);

– *function importSysKey(imKeyAlgoritm, imKeyLength, imKeyExport, clKeyAlgoritm)* – импорт симметричного ключа из файла и проверка наличия импортированного ключа в устройстве (*imKeyAlgoritm* – алгоритм импортируемого ключа, *imKeyLength* – длина импортированного ключа, *imKeyExport* – экспортируемость импортируемого ключа, *clKeyAlgoritm* – алгоритм закрытого ключа, с помощью которого выполняется импорт);

– *function deleteSymKey(keyAlgoritm, keyLength, keyExport)* – удаление симметричного ключа (*keyAlgoritm* – алгоритм ключа, *keyLength* – длина ключа, *keyExport* – экспортируемость ключа);

– *function nonExpSymKey(keyAlgoritm, keyLength, keyExport)* – проверка невозможности экспорта неэкспортируемого симметричного ключа (*keyAlgoritm* – алгоритм ключа, *keyLength* – длина ключа, *keyExport* – экспортируемость ключа);

– *function exportOpenKey(keyAlgoritm, keyLength, keyExport)* – экспорт открытого ключа ключевой пары (*keyAlgoritm* – алгоритм ключа, *keyLength* – длина ключа, *keyExport* – экспортируемость ключа);

– *function deleteOpenKey(keyAlgoritm, keyLength, keyExport)* – удаление открытого ключа ключевой пары (*keyAlgoritm* – алгоритм ключа, *keyLength* – длина ключа, *keyExport* – экспортируемость ключа);

– *function importOpenKey(keyAlgoritm, keyLength, keyExport)* – импорт открытого ключа ключевой пары (*keyAlgoritm* – алгоритм ключа, *keyLength* – длина ключа, *keyExport* – экспортируемость ключа);

– *function encFile(keyAlgoritm, keyLength, keyExport)* – зашифрование файла (*keyAlgoritm* – алгоритм ключа, *keyLength* – длина ключа, *keyExport* – экспортируемость ключа);

– *function decFile(keyAlgoritm, keyLength, keyExport)* – расшифрование файла и проверка содержимого с эталонным значением (*keyAlgoritm* – алгоритм ключа, *keyLength* – длина ключа, *keyExport* – экспортируемость ключа);

– *function checkDeleteEncDecFile()* – проверка удаления исходного файла при зашифровании и расшифровании;

– *function signFile(keyAlgoritm, keyLength)* – генерация ЭП файла (*keyAlgoritm* – алгоритм ключа, *keyLength* – длина ключа);

– *function checkSignFile(keyAlgoritm, keyLength)* – проверка ЭП файла (*keyAlgoritm* – алгоритм ключа, *keyLength* – длина ключа);

– *function checkModifSign()* – проверка некорректности ЭП при изменении файла подписи;

– *function checkModifOrigFile()* – проверка некорректности ЭП при изменении исходного файла;

– *function deleteAsymKeyAll(keyAlgoritm, keyLength, keyExport)* – удаление всех ключевых пар по заданным параметрам (*keyAlgoritm* – алгоритм ключа, *keyLength* – длина ключа, *keyExport* – экспортируемость ключа);

– *function checkExpSym(keyAlgoritm, keyLength)* – проверка невозможности экспорта экспортируемого симметричного ключа без использования защищенного алгоритма экспорта (*keyAlgoritm* – алгоритм ключа, *keyLength* – длина ключа);

– *function deletePrivKey(keyAlgoritm, keyLength, keyExport)* – удаление закрытого ключа ключевой пары (*keyAlgoritm* – алгоритм ключа, *keyLength* – длина ключа, *keyExport* – экспортируемость ключа).

Приложение Б

Перенаправление в виртуальную машину аппаратной компоненты стационарных программно-аппаратных СЗИ и используемых ими мобильных аппаратных идентификаторов

Привязка аппаратной компоненты с PCI-интерфейсом подключения выполняется по известным идентификационным номерам производителя и устройства (VID и PID) к драйверу pci_stub или vfio-pci следующим образом:

```
BASE_ADDRESS="0000"
PRODUCT_ID="03:00.0"
VENDOR_ID="1795 0700"

modprobe pci_stub # если ядро Linux младше версии 4.1
echo "${BASE_ADDRESS}:${PRODUCT_ID}" > /sys/bus/pci/devices/
  ${BASE_ADDRESS}:${PRODUCT_ID}/driver/unbind
echo "${VENDOR_ID}" > /sys/bus/pci/drivers/pci-stub/new_id
echo "${BASE_ADDRESS}:${PRODUCT_ID}" > /sys/bus/pci/drivers/pci-stub/bind

#modprobe vfio-pci # если ядро Linux старше версии 4.1
#echo "${BASE_ADDRESS}:${PRODUCT_ID}" > /sys/bus/pci/devices/ ${
   BASE_ADDRESS}:${PRODUCT_ID}/driver/unbind
#echo "${VENDOR_ID}" > /sys/bus/pci/drivers/vfio-pci/new_id
```

Перенаправление дополнительных мобильных аппаратных компонент, описываемых по известным идентификационным номерам производителя и устройства (VID и PID) или по расположению на соответствующей используемому интерфейсу шине в отдельном xml-файле описания *dst-hardware-interface-passthrough.xml*:

```
<hostdev mode='subsystem' type='interface' managed='no'>
  <source>
    <vendor id='0xa420'/>
    <product id='0x5426'/>
  </source>
</hostdev>
<hostdev mode='subsystem' type='interface' managed='no'>
  <source>
    <address bus='001' device='003'/>
  </source>
</hostdev>
```

где *interface* – название интерфейса дополнительной мобильной аппаратной компоненты СЗИ, например, *usb* для аппаратных идентификаторов с USB-интерфейсом подключения.

Приложение В

Фрагменты листингов разработанных программ тестирования, результатов их запуска и работы

Фрагмент программы тестирования функций безопасности программно-аппаратных СЗИ, реализованных на базе мобильной аппаратной компоненты в ОС СВТ, для ПСКЗИ ШИПКА (KeysEncSign) приведен в Листинге В.1. Фрагменты функций обработки ошибок в ходе работы этой программы тестирования и возврата программной и аппаратной компонент ПСКЗИ ШИПКА к начальному состоянию приведены в Листинге В.2.

Результаты запуска и работы программ тестирования функций безопасности программно-аппаратных СЗИ, реализованных на базе стационарной аппаратной компоненты до загрузки и в ОС СВТ, для ПАК СЗИ НСД «Аккорд-Х» приведены в Листинге В.3.

```
// Основная функция автоматизированного тестирования Ключи. Шифрование и подпись
   файлов
function KeysEncSign() {
  var pathToDesktop;

  if (aqEnvironment.GetWinMajorVersion() == 6)
    pathToDesktop = aqEnvironment.GetEnvironmentVariable("USERPROFILE") + "
   \\Desktop\\";
  else
    pathToDesktop = aqEnvironment.GetEnvironmentVariable("USERPROFILE") + "
   Рабочий\\ стол\\";

  // Подключение ПСКЗИ ШИПКА, включение коммутатора
  Sys.OleObject("WScript.Shell").Run('USBSwitcherConsole.exe on "OKB SAPR
   USB Switcher. 0"');

  TestedApps.ACshEncSig.Run();

  Log.AppendFolder("Вывод окна для ввода PIN при генерации ключа в устройстве,
   если PIN не был введен ранее", lmWarning);
  Log.PopLogFolder();

  //Открытие окна управления ключами
  Aliases.ACshEncSig.FileViewForm.VCLObject("tlbtnKeyManage").Click();

  //Открытие окна генерации симметричного ключа
  Aliases.ACshKeyManager.KeyDirViewForm.tlbtnGenKey.Click();

```

```
24  ...
25
26  Log.AppendFolder("Генерация симметричного ключа ГОСТ 28147-89 (256 бит,
     экспортируемый)");
27  //Выбор нужных параметров
28  Aliases.ACshKeyManager.GenKeyForm.cbAlgorithm.Keys("[Down][Down][Down]");
29  genSymKey(Aliases.ACshKeyManager.GenKeyForm, "ГОСТ 28147-89", "256", "
     экспортируемый");
30  Log.PopLogFolder();
31
32  Log.AppendFolder("Генерация симметричного ключа ГОСТ 28147-89 (256 бит, не
     экспортируемый)");
33  Aliases.ACshKeyManager.GenKeyForm.cbAlgorithm.Keys("[Down][Down][Down]");
34  Aliases.ACshKeyManager.GenKeyForm.cbExport.Keys("[Down]");
35  genSymKey(Aliases.ACshKeyManager.GenKeyForm, "ГОСТ 28147-89", "256", "
     неэкспортируемый");
36  Log.PopLogFolder();
37
38  //Закрытие окна генерации симметричных ключей
39  Aliases.ACshKeyManager.GenKeyForm.btbtnCancel.Click();
40
41  // Переподключение ПСКЗИ ШИПКА, выключение и включение коммутатора
42  Sys.OleObject("WScript.Shell").Run('USBSwitcherConsole.exe off "OKB SAPR
     USB Switcher. 0"');
43  Sys.OleObject("WScript.Shell").Run('USBSwitcherConsole.exe on "OKB SAPR
     USB Switcher. 0"');
44
45  //Открытие окна генерации ключевых пар
46  Aliases.ACshKeyManager.KeyDirViewForm.tlbtnGenKeyPair.Click();
47
48  ...
49
50  Log.AppendFolder("Генерация ключевой пары ГОСТ Р 34.10-2001 (512 бит,
     экспортируемый)");
51  Aliases.ACshKeyManager.GenKeyPairForm.cbAlgorithm.Keys("[Down]");
52  genAsyKey(Aliases.ACshKeyManager.GenKeyPairForm, "ГОСТ Р 3410-2001", "512
     ", "экспортируемый");
53  Log.PopLogFolder();
54
55  Log.AppendFolder("Генерация ключевой пары ГОСТ Р 34.10-2001 (512 бит,
     неэкспортируемый)");
56  Aliases.ACshKeyManager.GenKeyPairForm.cbAlgorithm.Keys("[Down]");
57  Aliases.ACshKeyManager.GenKeyPairForm.cbExport.Keys("[Down]");
```

```
genAsyKey(Aliases.ACshKeyManager.GenKeyPairForm, "ГОСТ Р 3410-2001", "512
 ", "неэкспортируемый", 2);
Log.PopLogFolder();

//Закрытие окна генерации ключевых пар
Aliases.ACshKeyManager.GenKeyPairForm.btbtnCancel.Click();

// Переподключение ПСКЗИ ШИПКА, выключение и включение коммутатора
Sys.OleObject("WScript.Shell").Run('USBSwitcherConsole.exe off "OKB SAPR
 USB Switcher. 0"');
Sys.OleObject("WScript.Shell").Run('USBSwitcherConsole.exe on "OKB SAPR
 USB Switcher. 0"');

...

  Log.AppendFolder("Экспорт симметричного ключа ГОСТ 28147-89 (256 бит,
 экспортируемый), используя чужой открытый ключ ГОСТ Р 34.10-2001, свой
 закрытый ключ ГОСТ Р 34.10-2001 ");
  exportSymKey("ГОСТ 28147-89", "256", "экспортируемый", "ГОСТ Р 3410-2001"
 );
  Log.PopLogFolder();
  Log.AppendFolder("Удаление симметричного ключа ГОСТ 28147-89, 256");
  deleteSymKey("ГОСТ 28147-89", "256", "экспортируемый");
  Log.PopLogFolder();
  Log.AppendFolder("Импорт симметричного ключа ГОСТ 28147-89, 256 на ГОСТ Р
 3410-2001");
  importSysKey("ГОСТ 28147-89", "256", "экспортируемый", "ГОСТ Р 3410-2001"
 );
  Log.PopLogFolder();
  Log.PopLogFolder();

// Переподключение ПСКЗИ ШИПКА, выключение и включение коммутатора
Sys.OleObject("WScript.Shell").Run('USBSwitcherConsole.exe off "OKB SAPR
 USB Switcher. 0"');
Sys.OleObject("WScript.Shell").Run('USBSwitcherConsole.exe on "OKB SAPR
 USB Switcher. 0"');

  ...

  Log.AppendFolder("Экспорт открытого ключа ключевой пары ГОСТ Р 3410-2001,
 512");
  exportOpenKey("ГОСТ Р 3410-2001", "512", "экспортируемый");
Log.PopLogFolder();
```

```
90  Log.AppendFolder("Удаление открытого ключа ключевой пары ГОСТ Р 3410-2001, 512
     ");
91  deleteOpenKey("ГОСТ Р 3410-2001", "512", "экспортируемый");
92  Log.PopLogFolder();
93  Log.AppendFolder("Импорт открытого ключа ключевой пары ГОСТ Р 3410-2001, 512"
     );
94  importOpenKey("ГОСТ Р 3410-2001", "512", "экспортируемый");
95  Log.PopLogFolder();
96
97  Log.AppendFolder("Вывод ошибки при попытке импорта ключа, который уже
     существует в устройстве");
98  Aliases.ACshKeyManager.KeyDirViewForm.tlbtnImportKey.Click();
99  Sys.Process("ACshKeyManager").Window("TbsOpenDlgForm", "", 1).Window("
     TbsSkinPanel", "", 1).Window("TbsSkinFileListView", "", 1).SelectItem("
     publickey.bin");
100 Aliases.ACshKeyManager.TbsOpenDlgForm2.TbsSkinPanel.TbsSkinButton.Click()
     ;
101 aqObject.CheckProperty(Sys.Process("ACshKeyManager").Window("
     TbsMessageForm", "Информация", 1).VCLObject("Message"), "Caption",
     cmpContains, "Этот ключ уже существует в устройстве");
102 Sys.Process("ACshKeyManager").Window("TbsMessageForm", "Информация", 1).
     VCLObject("OK").Click();
103 Log.PopLogFolder();
104
105   //Закрытие окна управления ключами
106 Aliases.ACshKeyManager.KeyDirViewForm.Close();
107   //Переход к обзору рабочего стола в утилите
108 Aliases.ACshEncSig.FileViewForm.tlbtnUpOneLevel.Click();
109
110 // Переподключение ПСКЗИ ШИПКА, выключение и включение коммутатора
111 Sys.OleObject("WScript.Shell").Run('USBSwitcherConsole.exe off "OKB SAPR
     USB Switcher. 0"');
112 Sys.OleObject("WScript.Shell").Run('USBSwitcherConsole.exe on "OKB SAPR
     USB Switcher. 0"');
113
114 ...
115
116 Log.AppendFolder("Шифрование на ГОСТ 28147-89, 256, экспортируемый");
117 encFile("ГОСТ 28147-89", "256", "экспортируемый");
118 Log.PopLogFolder();
119 Log.AppendFolder("Расшифрование на ГОСТ 28147-89, 256, экспортируемый");
120 decFile("ГОСТ 28147-89", "256", "экспортируемый");
121 Log.PopLogFolder();
```

```
122
123 Log.AppendFolder("Шифрование на ГОСТ 28147-89, 256, неэкспортируемый");
124 encFile("ГОСТ 28147-89", "256", "неэкспортируемый");
125 Log.PopLogFolder();
126 Log.AppendFolder("Расшифрование на ГОСТ 28147-89, 256, неэкспортируемый");
127 decFile("ГОСТ 28147-89", "256", "неэкспортируемый");
128 Log.PopLogFolder();
129
130 Log.AppendFolder("Удаление исходного файла после завершения шифрования при
     выбранном параметре «Удалить исходный файл после шифрования» и «Удалить исходный
     файл после расшифрования»");
131 checkDeleteEncDecFile();
132 Log.PopLogFolder();
133
134 ...
135
136 // Переподключение ПСКЗИ ШИПКА, выключение и включение коммутатора
137 Sys.OleObject("WScript.Shell").Run('USBSwitcherConsole.exe off "OKB SAPR
     USB Switcher. 0"');
138 Sys.OleObject("WScript.Shell").Run('USBSwitcherConsole.exe on "OKB SAPR
     USB Switcher. 0"');
139
140 ...
141
142 Log.AppendFolder("Удаление всех ключевых пар из устройства");
143
144 ...
145
146 deleteAsymKeyAll("ГОСТ Р 3410-2001", "512", "экспортируемый");
147 deleteAsymKeyAll("ГОСТ Р 3410-2001", "512", "неэкспортируемый");
148 Log.PopLogFolder();
149
150 ...
151
152 Log.AppendFolder("Попытка экспорта симметричного ключа ГОСТ 28147-89 (256 бит,
     экспортируемый) по алгоритму ГОСТ Р 3410-2001, когда  в устройстве нет закрытого
     ключа ГОСТ Р 3410-2001");
153 checkExpSym("ГОСТ 28147-89", "256");
154 Log.PopLogFolder();
155
156 ...
157
158 Log.AppendFolder("Удаление закрытого ключа ГОСТ Р 3410-2001, 512");
```

```
  deleteAsymKeyAll("ГОСТ Р 3410-2001", "512", "экспортируемый");
  Log.PopLogFolder();

  ...

  //Закрытие окна генерации ключевых пар
  Aliases.AShKeyManager.GenKeyPairForm.btbtnCancel.Click();
  Log.PopLogFolder();

  ...

  // Отключение ПСКЗИ ШИПКА, выключение коммутатора
  Sys.OleObject("WScript.Shell").Run('USBSwitcherConsole.exe off "OKB SAPR
   USB Switcher. 0"');

}
```

Листинг В.1 – Фрагмент программы тестирования функций безопасности ПСКЗИ ШИПКА KeysEncSign

```
needRecovery = false;

// Функция перехвата событий записи ошибки в журнал
function GeneralEvents_OnLogError(Sender, LogParams) {
  // Обработка конкретных ошибок
  switch (LogParams.MessageText)
  {
    case 'Error1':
      ...
      Log.Message("Описание Error1");
      break;

    ...

    default:
      needRecovery = true;
      Log.Message("Отработало needRecovery()");
  }
}

...

```

```
// Обработка необходимости возврата ПСКЗИ ШИПКА к начальному состоянию перед выполнением очередной проверки
if (needRecovery)
{
  recoveryWork()
  needRecovery = false;
}
...

// Функция возврата ПСКЗИ ШИПКА к начальному состоянию
function recoveryWork() {
  // Перезапуск программной компоненты
  if (Aliases.ACshEncSig.Exists)
  {
    Aliases.ACshEncSig.Terminate();
    aqUtils.Delay(1500);
  }
  TestedApps.ACshEncSig.Run();

  // Возврат к начальному состоянию: форматирование устройства, удаление ключей и так далее
  ...
}
```

Листинг В.2 – Фрагменты функций обработки ошибок и возврата программной и аппаратной компонент ПСКЗИ ШИПКА к начальному состоянию

```
# Вывод списка ВМ, на которых проводится запуск программ тестирования
/# virsh --connect qemu:///system list --all
 Id    Name                           State
----------------------------------------------------
 1     acx-deploy                     running
 2     rhel5.7-i686                   running
 -     altlinux6.0.1.spt-i686         shut off
 -     altlinux7.0.5-i686             shut off
 -     altlinux7.0.5-x86_64           shut off
 -     astra1.3-x86_64                shut off
 -     debian7.6.0-x86_64             shut off
 -     rhel5.1-i686                   shut off
 -     rhel5.4-i686                   shut off
 -     rhel6.1-x86_64                 shut off
 -     rhel6.4-x86_64                 shut off
 -     rhel7.0-x86_64                 shut off
```

```
 -      rosa.cobalt1.0-i686              shut off
 -      rosa.cobalt1.0-x86_64            shut off
 -      ubuntu12.04.5-i686               shut off
 -      ubuntu12.04.5-x86_64             shut off

# Запуск проверок для АМДЗ
/# acxhost test-amdz rhel5.7-i686
Testing Accord-AMDZ on remote host:
Name: rhel5.7-i686. IP: 192.168.1.8

Starting...
Running tests...
# Первоначальная загрузка и инициализация
Running test: {First Boot Test}
# Базовая настройка комплекса
Running test: {Base Configuration Test}
Rebooting...
# Проверка функций идентификации и аутентификации
Running test: {Login Test}
# Предъявление соответствующего аппаратного идентификатора
  Waiting for HW identifiers...
Rebooting...
# Проверка функций идентификации и аутентификации с различными видами идентификаторов
Running test: {Login HW Identifiers Test}
# Предъявление соответствующих аппаратных идентификаторов
  Waiting for HW identifiers...
# Настройка функции контроля целостности программных и "аппаратных" компонент ВМ
Running test: {ICL Test}
Rebooting...
# Проверка функции контроля целостности программных и "аппаратных" компонент ВМ
Running test: {ICL Check Test}
Rebooting...
# Проверка корректности дальнейшей доверенной загрузки ОС ВМ
Running test: {OS Boot Test}

# Результаты тестирования функций безопасности АМДЗ
Tests finished. You can see results in /opt/acxhost/hosts/rhel5.7-i686/logs
   /amdz-tests directory.
Total test count:     56
Total failed tests:   0


# Запуск автоматической сборки подсистемы разграничения доступа на ВМ
```

```
/# acxhost build rhel5.7-i686
Building Accord-X on remote host:
Name: rhel5.7-i686. IP: 192.168.1.8

Preparing...
Configuring...
Building...
Building passed for time: 3m
Building distrib...
Building distrib passed for time: 55s
Successfully builded distrib, see the /opt/acxhost/hosts/rhel5.7-i686/rpms
    directory.


# Запуск автоматической установки и настройки подсистемы разграничения доступа в ОС ВМ
/# acxhost install rhel5.7-i686
Installing Accord-X on remote host:
Name: rhel5.7-i686. IP: 192.168.1.8

# Этап 1: установка пакетов, настройка компонент ОС и загрузчика, базовая настройка СЗИ
Installing Accord-X Stage 1
Installing packages...

Preparing...
   ##########################################################
acx-admin
   ##########################################################
AccordX security framework administration utilities installed successfully

Preparing...
   ##########################################################
acx-core-remote
   ##########################################################
AccordX security framework core with remote access installed successfully

Preparing...
   ##########################################################
acx-tmid-usb
   ##########################################################
TM-usb devices support software installed successfully

Preparing...
   ##########################################################
```

```
acx-amdz
   ############################################################
Accord-AMDZ devices support software installed successfully

Configuring initrd...
########################################################################

Configuring grub...
########################################################################

Configuring pam...
########################################################################

Configuring Accord-X...
# Предъявление соответствующего аппаратного идентификатора
  Waiting for HW identifiers...
Rebooting...

# Этап 2: проверка подсистемы разграничения доступа с базовыми настройками
Installing Accord-X Stage 2 (base tests)
# Предъявление соответствующих аппаратных идентификаторов
  Waiting for HW identifiers...
Installing Accord-X Successfully ended!


# Запуск программ тестирования подсистемы разграничения доступа в ОС ВМ
/# acxhost test rhel5.7-i686
Testing Accord-X on remote host:
Name: rhel5.7-i686. IP: 192.168.1.8

Configuring...
Rebooting...

# Генерация объектов и правил с различным параметрами для проверок
Generating tests...
Reboot, waiting for ssh connection...

Running tests...
# Проверка функций идентификации и аутентификации
Running test: {Login Test}
# Предъявление соответствующего аппаратного идентификатора
  Waiting for HW identifiers...
# Проверка функций идентификации и аутентификации с различными видами идентификаторов
```

```
136 Running test: {Login HW Identifiers Test}
137 # Предъявление соответствующих аппаратных идентификаторов
138   Waiting for HW identifiers...
139 # Проверка функций идентификации и аутентификации при удаленном входе в ОС
140 Running test: {Remote Login Test}
141 # Предъявление соответствующего аппаратного идентификатора
142   Waiting for HW identifiers...
143 # Проверка атрибутов на открытие объектов при дискреционном управлении доступом
144 Running test: {Open Test}
145 # Проверка атрибутов на создание, удаление и переименование
146 Running test: {Create, Delete and Rename File Test}
147 # Проверка атрибутов на создание жестких ссылок
148 Running test: {Link Test}
149 # Проверка разграничения доступа к каталогам
150 Running test: {Directory Rights Test}
151 # Проверка разграничения доступа с учетом рекурсии
152 Running test: {Directory Recursion Test}
153 # Проверка правил мандатной политики управления доступом
154 Running test: {Mandate ACL Files Test}
155 # Проверка правил мандатной политики управления доступом для каталогов
156 Running test: {Mandate ACL Directories Test}
157 # Проверка динамического контроля целостности
158 Running test: {Dynamic ICL Test}
159 # Проверка различных способов осуществления доступа субъектов к объектам
160 Running test: {Alternate Access Test}
161
162 # Результаты тестирования функций безопасности подсистемы разграничения доступа
163 Tests finished. You can see results in /opt/acxhost/hosts/rhel5.7-i686/logs
      /tests directory.
164 Total test count:     88120
165 Total failed tests:   3
166
167 # Выявленные в ходе тестирования ошибки с дополнительной информацией
168 Failed tests:
169 [Test Failed] <File deletion check> G /TestDiscr/TestDirRecursionS/G/G/EDG/
      D_FILE
170 [Test Failed] <Mand rename check> 3 /TestMand/TestMandFiles/
      TEST_FILE_MAND_4456_LEVEL_3
171 [Test Failed] <Dynamic icl check> DYNAMIC /TestIcl/DynamicIclTest/
      DYNAMIC_TEST_N4
```

Листинг В.3 – Запуск и результаты работы программ тестирования ПАК СЗИ НСД «Аккорд-Х» на одной из ВМ

Приложение Г

Документы, подтверждающие внедрение результатов диссертации

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2016616332

Тестирование функций безопасности программно-аппаратных средств защиты информации

Правообладатель: ***Закрытое акционерное общество "Особое Конструкторское Бюро Систем Автоматизированного Проектирования" (RU)***

Авторы: ***Каннер Татьяна Михайловна (RU), Коробов Владимир Владимирович (RU)***

Заявка № **2016613652**
Дата поступления **14 апреля 2016 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***09 июня 2016 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

ОКБ САПР ОСОБОЕ КОНСТРУКТОРСКОЕ БЮРО
СИСТЕМ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ

ЗАО «ОКБ САПР» 115114, г. Москва, 2-й Кожевнический переулок, 12 Тел. +7 (495) 994-72-62
ИНН 7725739137, КПП 772501001 okbsapr@okbsapr.ru, www.okbsapr.ru

«УТВЕРЖДАЮ»
Генеральный директор
ЗАО «ОКБ САПР»,
к.т.н.

_____ И.Г. Назаров
«__» _____ 2021 г.

АКТ
о практическом применении (внедрении) результатов диссертационного исследования Каннер Т.М. на тему «Моделирование состояний аппаратной компоненты для тестирования средств защиты информации»

Настоящим комиссия в составе заместителя директора Счастного Д.Ю., начальника отдела программирования средств защиты информации (СЗИ) Мозолиной Н.В., руководителя группы программирования ПО СЗИ Чадова А.Ю. удостоверяет, что следующие результаты проведенного Каннер Т.М. диссертационного исследования на тему «Моделирование состояний аппаратной компоненты для тестирования средств защиты информации»:

- модель программно-аппаратных СЗИ, реализующих подлежащие тестированию функции безопасности, учитывающая состояния их аппаратной компоненты;
- критерии применимости существующих способов тестирования к функциям безопасности программно-аппаратных СЗИ;
- алгоритм тестирования функций безопасности программно-аппаратных СЗИ;
- алгоритм верификации программно-аппаратных СЗИ;
- способ тестирования программно-аппаратных СЗИ, учитывающий состояния их аппаратной компоненты;
- программно-аппаратный комплекс тестирования функций безопасности программно-аппаратных СЗИ

применяются для тестирования программно-аппаратных комплексов средств защиты информации, производимых компанией «ОКБ САПР» и

встраиваемых в различные информационные системы, а также в рамках разработки собственных перспективных средств тестирования.

В большинстве известных работ российских и зарубежных ученых рассматриваются вопросы тестирования только программного обеспечения. При этом выполненное исследование позволяет решить актуальный вопрос применимости существующих способов и средств тестирования в отношении функций безопасности программно-аппаратных средств защиты информации. В связи с этим, по заключению комиссии, исследование Каннер Т.М. отвечает требованиям, предъявляемым к научным работам, а созданное на его основе программно-аппаратное решение позволяет в значительной степени сократить время разработки и тестирования продуктов компании, существенно уменьшить возможные риски нарушения работы использующих их информационных систем с повышением общего уровня защищенности.

Председатель комиссии:

Зам. генерального директора _____ Д.Ю. Счастный

Члены комиссии:

Начальник отдела
программирования СЗИ _____ Н.В. Мозолина

Руководителя группы
программирования ПО СЗИ _____ А.Ю. Чадов

ФЕДЕРАЛЬНОЕ
АВТОНОМНОЕ УЧРЕЖДЕНИЕ
«ГОСУДАРСТВЕННЫЙ
НАУЧНО-ИССЛЕДОВАТЕЛЬСКИЙ
ИСПЫТАТЕЛЬНЫЙ ИНСТИТУТ ПРОБЛЕМ
ТЕХНИЧЕСКОЙ ЗАЩИТЫ ИНФОРМАЦИИ
ФЕДЕРАЛЬНОЙ СЛУЖБЫ
ПО ТЕХНИЧЕСКОМУ
И ЭКСПОРТНОМУ КОНТРОЛЮ»
(ФАУ «ГНИИИ ПТЗИ ФСТЭК РОССИИ»)

Студенческая ул., д. 36, г. Воронеж, 394036
Тел., факс: (473) 257-92-58, 279-25-16
E-mail: gniii@fstec.ru

10 02 2021 №

На №

УТВЕРЖДАЮ
Начальник ФАУ «ГНИИИ ПТЗИ
ФСТЭК России»
д-р техн. наук, ст. науч. сотр.

А.В. Анищенко
«10» 02 2021 г.

Акт

о внедрении результатов диссертационной работы Каннер Т.М. на тему «Моделирование состояний аппаратной компоненты для тестирования средств защиты информации» в Федеральном автономном учреждении «Государственный научно-исследовательский институт проблем технической защиты информации Федеральной службы по техническому и экспортному контролю»

Комиссия в составе кандидата технических наук Смолина А.В., кандидата технических наук Каданцева И.А. и Короткова Д.В. составила настоящий акт о том, что результаты диссертационной работы Каннер Т.М., связанные с тестированием программно-аппаратных средств защиты информации, а именно:

– алгоритмы тестирования и верификации функций безопасности программно-аппаратных средств защиты информации;

– способ тестирования средств защиты информации, учитывающий состояния их аппаратной компоненты,

используются в ФАУ «ГНИИИ ПТЗИ ФСТЭК России» при разработке программ и методик сертификационных испытаний и верификации результатов тестирования сертифицируемых средств защиты информации.

Применение указанного способа и алгоритмов позволило существенно снизить временные издержки на трудозатратный процесс тестирования средств защиты информации и удостовериться в отсутствии негативного влияния сертифицируемых средств защиты на характеристики информационных систем.

Председатель комиссии
начальник Центра
кандидат техн. наук А.В. Смолин

Заместитель начальника Центра
кандидат техн. наук И.А. Каданцев

Старший научный сотрудник Д.В. Коротков

МИНИСТЕРСТВО
НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский ядерный университет «МИФИ»
(НИЯУ МИФИ)
Каширское шоссе, д.31, г. Москва, 115409
Тел. (499) 324-77-77, факс (499) 324-21-11
http://www.mephi.ru

17.02.2021 № 042/01

На № _____ от _____

«УТВЕРЖДАЮ»
Проректор НИЯУ МИФИ
_____Н.И. Каргин
«17» 02 2021 г.

Акт

об использовании результатов диссертационной работы Каннер Татьяны Михайловны на тему «Моделирование состояний аппаратной компоненты для тестирования средств защиты информации»

СОСТАВ КОМИССИИ:

Председатель комиссии

И.о. заведующего кафедрой «Стратегические информационные исследования», к.т.н., доцент А.П. Дураковский

Члены комиссии

Доцент отделения интеллектуальных кибернетических систем офиса образовательных программ (М), к.т.н., доцент В.С. Горбатов;

Ассистент отделения интеллектуальных кибернетических систем офиса образовательных программ (М) Г.П. Гавдан.

Комиссия составила настоящий акт о том, что результаты диссертационной работы Каннер Т.М. на тему «Моделирование состояний аппаратной компоненты для тестирования средств защиты информации»:

1. алгоритмы тестирования и верификации функций безопасности программно-аппаратных средств защиты информации;

2. способ тестирования программно-аппаратных СЗИ, учитывающий состояния аппаратной компоненты;
3. программно-аппаратный комплекс тестирования СЗИ

обладают актуальностью, представляют практический интерес, были изучены и использованы нами при создании инженерно-технических решений для высокотехнологического производства инновационных программно-аппаратных средств защиты информации на базе перспективных высокоскоростных интерфейсов информационного взаимодействия в рамках исполнения работ по НИОКТР «Создание инженерно-технических решений для высокотехнологичного производства инновационных программно-аппаратных средств защиты информации на базе перспективных высокочастотных интерфейсов информационного взаимодействия».

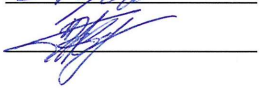
Проведенное диссертантом исследование послужило теоретическим фундаментом и, во многом, практическим руководством для успешного выполнения необходимых этапов работ по НИОКТР.

Председатель комиссии:

А.П. Дураковский

Члены комиссии:

В.С. Горбатов



Г.П. Гавдан

«УТВЕРЖДАЮ»
Проректор
НИЯУ МИФИ
_____ Е.Б. Весна
«__» _____ 2021 г.

А К Т

**о внедрении результатов диссертационной работы Каннер Т.М.
на тему «Моделирование состояний аппаратной компоненты для тестирования средств защиты информации»
в курс «Программно-аппаратные средства защиты информации»
кафедры «Криптология и кибербезопасность» (№ 42) НИЯУ МИФИ**

Комиссия в составе: зам. заведующего кафедрой № 42 к.т.н. Когоса К.Г., д.т.н., проф. Иваненко В.Г., д.т.н., доцента Запечникова С.В. составила настоящий акт о том, что результаты диссертационной работы Каннер Т.М., связанные с тестированием программно-аппаратных средств защиты информации, а именно:

– анализ существующих способов тестирования программного обеспечения для проверки функций безопасности программно-аппаратных средств защиты информации;
– модель программно-аппаратных СЗИ, реализующих подлежащие тестированию функции безопасности, учитывающая состояния их аппаратной компоненты;
– алгоритмы тестирования и верификации функций безопасности программно-аппаратных средств защиты информации;
– способ тестирования программно-аппаратных СЗИ, учитывающий состояния аппаратной компоненты

используются при чтении лекций и проведении лабораторных занятий по курсу «Программно-аппаратные средства защиты информации» на кафедре «Криптология и кибербезопасность» института интеллектуальных кибернетических систем НИЯУ МИФИ.

Применение приведенных выше результатов диссертационной работы позволяет слушателям данного учебного курса ознакомиться с практическим использованием современных методов тестирования программного-аппаратных средств защиты информации.

_____ к.т.н. Когос К.Г.
_____ д.т.н. Иваненко В.Г.
_____ д.т.н. Запечников С.В.